digital

Digital Semiconductor Alpha Microprocessors SROM Mini-Debugger

User's Guide

Order Number: EC–QHUXC–TE

Revision/Update Information: This is a revised document. It supersedes the *Digital Semiconductor Alpha Microprocessors SROM Mini-Debugger User's Guide* (EC–QHUXB–TE).

Digital Equipment Corporation
Maynard, Massachusetts

http://www.digital.com/semiconductor

May 1997

While Digital believes the information included in this publication is correct as of the date of publication, it is subject to change without notice.

Digital Equipment Corporation makes no representations that the use of its products in the manner described in this publication will not infringe on existing or future patent rights, nor do the descriptions contained in this publication imply the granting of licenses to make, use, or sell equipment or software in accordance with the description.

©Digital Equipment Corporation 1997. All rights reserved.
Printed in U.S.A.

Digital, Digital Semiconductor, and the DIGITAL logo are trademarks of Digital Equipment Corporation.

Digital Semiconductor is a Digital Equipment Corporation business.

Intel is a registered trademark of Intel Corporation.
Windows NT is a trademark and Microsoft is a registered trademark of Microsoft Corporation.

All other trademarks and registered trademarks are the property of their respective owners.

Contents

Preface

1 Introduction

1.1 Overview 1–1
1.2 General Features 1–1

2 Getting Started

2.1 Overview 2–1
2.2 Hardware Required 2–1
2.3 Hardware Debug Features 2–2
2.4 Setting Up the SROM Serial Port Connection 2–2
2.4.1 Connecting the Motherboard to a Terminal 2–2
2.4.2 Connecting the Motherboard to a Personal Computer 2–3
2.4.3 Connecting to the Motherboard from a System Running Windows NT Version 3.51 or Earlier 2–3
2.4.4 Connecting to the Motherboard from a System Running Windows NT Version 4.0 2–4
2.4.5 Connecting to the Motherboard from a System Running DIGITAL UNIX 2–4
2.4.5.1 Connecting to a Serial Port Under DIGITAL UNIX 2–5
2.5 Starting and Running the Mini-Debugger 2–6
2.5.1 Default Conditions 2–6
2.6 Sample Session on the EB64 2–7
2.6.1 EB64 Sample Log File 2–8
2.7 Sample Session on the EB64+ and the AlphaPC 64 2–10
2.7.1 EB64+ Sample Log File 2–11
2.8 Sample Session on the EB66 and EB66+ 2–14
2.8.1 EB66 Sample Log File 2–17
2.9 Sample Session on the EB164 2–21
2.9.1 EB164 Sample Log File 2–23
2.10 Sample Session on the AlphaPC 164 2–28
2.10.1 AlphaPC 164 Sample Log File 2–31
2.11 Sample Session on the AlphaPC 164LX 2–38
2.11.1 AlphaPC 164LX Sample Log File 2–39
2.12 Onboard Machine Check Handler 2–45

3 SROM Mini-Debugger Command Set

3.1 Overview 3–1
3.2 Command and User Interface Features 3–1
3.3 Command Summary 3–2
3.4 User Commands 3–3
!d 3–4
ba 3–6
bm 3–8
cm 3–9
cp 3–11
dc 3–12
di 3–14
dm 3–16
ec 3–18
em 3–21
fl 3–22
fm 3–24
fr 3–27
fw 3–31
lo 3–33
mt 3–35
pr 3–37
qw 3–39
rt 3–40
sb 3–41
st 3–43
wa 3–44
xb 3–47
xm 3–48

A Support, Products, and Documentation

Index

Tables

3–1 Command Summary 3–2
3–2 CPU Register Names for the dc Command 3–12
3–3 CPU Register Names for the ec Command 3–18

Preface

Introduction

This document describes how to use the Alpha Microprocessors SROM Mini-Debugger (also referred to as the mini-debugger) to debug hardware with one of the following evaluation boards:

- The AlphaPC 164LX Motherboard (AlphaPC 164LX)
- The AlphaPC 164 Motherboard (AlphaPC 164)
- The Alpha 21164 Evaluation Board (EB164)
- The AlphaPC 64 Motherboard (AlphaPC 64)
- The Alpha 21066A Evaluation Board (EB66+)
- The Alpha 21064 and 21064A PCI Evaluation Board (EB64+)
- The Alpha 21066 and 21068 Evaluation Board (EB66)
- The Alpha 21064 Evaluation Board (EB64)

Audience

This document is intended for anyone who develops hardware or software to be used with an Alpha microprocessor.

Content Overview

The information in this document is organized as follows:

- Chapter 1 is a general overview of the mini-debugger.
- Chapter 2 describes how to set up and start the mini-debugger.
- Chapter 3 describes the mini-debugger command set.
- Appendix A contains information about technical support services, products, and associated documentation.

Conventions

The following conventions are used in this document:

Convention	Meaning
A percent sign (%)	Indicates a DIGITAL UNIX operating system command prompt.
SROM>	Indicates an Alpha Microprocessors SROM Mini-Debugger prompt.
Boldface type	Indicates an Alpha Microprocessors SROM Mini-Debugger command keyword.
Italic type	Indicates special emphasis or the title of a manual.
`Monospaced type`	Indicates an operating system command, a file name, or a directory path name.
Underlined text	Indicates user input in an example.
Note	Provides general information.

1
Introduction

1.1 Overview

The Alpha Microprocessors SROM Mini-Debugger provides basic hardware debugging capability through the SROM serial port of the Alpha microprocessor. Using only an SROM containing the mini-debugger, a clock source, a CPU chip, and a few gates, you can exercise the device connected to the CPU to debug cache, memory, and I/O subsystems until the board is functional enough to support a more fully featured monitor.

1.2 General Features

The mini-debugger has the following features:

- Basic hardware debugging capability
- A monitor that can point to hardware addresses and exercise them
- The ability to examine and deposit memory
- A case-independent command language
- Support for variable baud rates and processor speeds

2
Getting Started

2.1 Overview

The Alpha Microprocessors Motherboard Software Design Tool Kit (Alpha SDK) includes the Alpha Microprocessors SROM Mini-Debugger binary files suitable for programming an SROM. For information about how to program an SROM, refer to your ROM programmer manual.

After the SROM is programmed, it can be placed into the SROM socket on the motherboard. In addition, the mini-debugger is also available in the standard SROM provided with the Alpha microprocessor motherboards. It can be invoked *after* the standard SROM has completed CPU and system initialization and before it begins execution of the image loaded from ROM. Refer to the Alpha microprocessor motherboard user's guide for more information about how to access the mini-debugger. It usually requires setting a jumper.

2.2 Hardware Required

To run the mini-debugger, you need the following items:

- An Alpha microprocessor motherboard or a system based on the Alpha microprocessor architecture with a connection from the microprocessor SROM interface to the host system's serial port (for example, an RS232)
- A host system (a terminal or workstation)
- An SROM containing the SROM Mini-Debugger image

2.3 Hardware Debug Features

The mini-debugger image is loaded into the CPU's instruction cache at reset through the CPU's SROM interface. The mini-debugger provides commands to:

- Examine and deposit data in memory.
- Test memory.
- Examine and deposit internal CPU registers.
- Load an image into the motherboard's memory and transfer execution to it.

The mini-debugger's primary purpose is to debug hardware so that the memory interface works, thus allowing a more complex debugger such as the Alpha Microprocessors Debug Monitor to be loaded to debug other parts of the system or software.

2.4 Setting Up the SROM Serial Port Connection

To use the mini-debugger, you must first establish a connection from your Alpha microprocessor system or motherboard to the serial port on your host system. This section describes how to connect the SROM serial port of an motherboard to the following hardware:

- A terminal
- A PC running communication software
- A system running Windows NT
- An Alpha system running DIGITAL UNIX

2.4.1 Connecting the Motherboard to a Terminal

To connect a motherboard to a terminal, connect the SROM serial port of the motherboard to the terminal communication line.

Set terminal settings as shown in the following table:

Terminal Setting	Value
Terminal emulation	VT100
Transmit/receive speed	9600 baud
Data bits	8
Parity	None
Stop bits	1

2.4.2 Connecting the Motherboard to a Personal Computer

You can also use communication (terminal emulation) software running on a PC to communicate with the motherboard. To connect the motherboard to a PC, connect the terminal communication line to the SROM serial port of the motherboard as described for the terminal.

2.4.3 Connecting to the Motherboard from a System Running Windows NT Version 3.51 or Earlier

A system running the Windows NT version 3.51 or earlier operating system supports serial communication with the motherboard. To configure a COM port for connection to the motherboard, follow these steps:

1. Choose the `Program Manager` icon.
2. Choose the `Accessories` icon.
3. Choose the `Terminal` icon.
4. Set the following terminal characteristics:

Terminal Setting	Value
Data bits	8 bit
Transmit/receive speed	9600 baud
Character format	No parity
Stop bits	1

Save these settings in a file. For example, settings for the EB64 could be saved in a file called `eb64.trm`.

All examples and command descriptions that follow assume that the motherboard SROM port is connected to COM1.

2.4.4 Connecting to the Motherboard from a System Running Windows NT Version 4.0

A system running the Windows NT version 4.0 operating system supports serial communication with the motherboard. Use the Start button on the taskbar to configure a COM port for connection to the motherboard and follow these steps:

1. Choose the `Programs` menu.
2. Choose the `Accessories` menu.
3. Choose the `HyperTerminal` menu and choose the `HyperTeminal` icon. The Connection Description window appears.
4. Enter a name for the new connection such as `Direct Connection Com1`, choose an icon for the connection, and click OK.
5. Set the following terminal characteristics:

Terminal Setting	Value
Bits per second	9600
Data bits	8
Parity	None
Stop bits	1
Flow control	None

Click OK to save these settings. The motherboard connection appears in the HyperTerminal window.

2.4.5 Connecting to the Motherboard from a System Running DIGITAL UNIX

An Alpha system running the DIGITAL UNIX operating system supports serial communication through two ports that can be connected to the motherboard:

- /dev/tty00
- /dev/tty01

All examples and command descriptions that follow assume that the motherboard SROM port is connected to port `/dev/tty00`.

To enable these ports for use with the motherboard, follow these steps:

1. Log in as superuser.
2. Modify the following two files:

   ```
   /etc/remote
   /etc/inittab
   ```

   a. Add the following two lines to the `/etc/remote` file. These lines define a device to connect to when using the DIGITAL UNIX `tip` command.

   ```
   port_name0:dv=/dev/tty00:br#9600:pa=none:
   port_name1:dv=/dev/tty01:br#9600:pa=none:
   ```

   The *port_name* refers to an arbitrary name that you assign to that port.

   b. Modify the `/etc/inittab` file to disable logins on the two serial communication ports by setting the third field to `off`. For example, modify the `tty00` and `tty01` lines as follows:

   ```
   tty00:23:off:/usr/sbin/getty /dev/tty00 9600
   tty01:23:off:/usr/sbin/getty /dev/tty01 9600
   ```

3. Reboot the system, or issue the following command to ensure that the modified files take effect:

   ```
   #  /sbin/init q
   ```

2.4.5.1 Connecting to a Serial Port Under DIGITAL UNIX

After you modify the `/etc/remote` and `/etc/inittab` files, you can connect to the serial port under the DIGITAL UNIX operating system using the DIGITAL UNIX `tip` command. If the connection is successful, the mini-debugger prompt displays after you press a key. For example:

```
% tip port_name0
                              ! key is pressed.
SROM>
```

Type ~. to exit the DIGITAL UNIX `tip` command.

2.5 Starting and Running the Mini-Debugger

After the SROM serial port connection has been made, you can initialize the mini-debugger by typing an ASCII character. This returns an SROM> prompt, which indicates that you are ready to begin debugging hardware and displays the mini-debugger version number.

For example:

```
U
V00000801
SROM>
```

Once an ASCII character is typed, the mini-debugger automatically detects the baud rate of the terminal connected to the SROM serial port. Baud rates up to 19.2K are supported.

2.5.1 Default Conditions

If you are using the mini-debugger built into the standard Alpha motherboard SROM, then the proper initialization conditions are automatically set.

The following sample sessions show you how to set up and run a standalone version of the mini-debugger. If you are using a standalone version of the mini-debugger, the following default conditions apply:

- Machine checks are disabled. (This does not apply to designs based on the Alpha 21164.)
- Data cache (Dcache) is off.
- The ABOX_CTL register is set to 0. (This does not apply to designs based on the Alpha 21164.)
- The ICCSR register is set to 000005F800000000 for boards based on the 21064 or 21066 and to 0xC24E000000 for boards based on the Alpha 21164.
- The pal_base is set to where the code is loaded; in this case, address 0.
- In boards based on the Alpha 21164, all three sets in the secondary cache are turned on and flushed. The block size has been set to 64 bytes.

Before you begin to debug hardware with the mini-debugger, you need to set several other registers depending on the microprocessor. See the sample sessions for more information.

2.6 Sample Session on the EB64

For the EB64 and the EB64+, the BIU_CTL register is cleared when you start the mini-debugger. Before any memory command can be executed on the EB64, the BIU_CTL register and the EB64 system register must be set to some meaningful value.

Note: To avoid damaging your hardware, bit 2 (OE) of the BIU_CTL is set at power-on and whenever the BIU_CTL register is written (from the **dc bctl** command).

To specify that the Bcache is *on*, use the following values:

```
BIU_CTL : 0000000E 2000E445
EB64 system register: address  00000002 00000000
                      data     00C40000 (for 32MB memory)
```

To specify that the Bcache is *off*, use the following values:

```
BIU_CTL : 0000000E 2000E444
EB64 system register: address  00000002 00000000
                      data     00C40000 (for 32MB memory)
```

You also need to initialize the clocks, timers, and memory refresh on the EB64. The following addresses should contain the specified data:

Address	Data
00000002 C0007600	00000001
00000002 C0007680	0000BF00
00000002 C0007600	00000006
00000002 C0007680	00000000
00000002 C0007600	00000005
00000002 C0007680	00000400
00000002 C0002180	74000000
00000002 C0002080	00001100
00000002 C0002080	00000000

2.6.1 EB64 Sample Log File

The following sample log file initializes the EB64.

```
       ! Sample EB64 Initialization Log file.

V00001c01
SROM> dc                          ! Deposit to the BIU_CTL register
IPR> bctl                         ! and turn off the BCache.
D> e2000e444
*BCTL   0000000e.2000e444

SROM> ec
Abox    00000000.00000000
Icsr    00000000.00bf0000
PalB    00000000.00000000
ExAd    00004000.00000002
DcSt    00000000.0000000f
Hirr    00000000.00000000
Hier    00000000.00000000
BCtl    0000000e.2000e444         ! This shows BIU_CTL has been changed.
BiSt    00000000.00003240
BiAd    00000000.28210b88
Syn     00000000.00000000
FiAd    00000000.28210b88

SROM> dm                          ! Turn off the BCache, set DRAM size to
A>200000000                       ! 2Mx36 (32MB of memory) and 1 cycle CAS
D> 2c40000                        ! slip in EB64 system register.

SROM> em                          ! Some bits in the system register are
A> 200000000                      ! Read-Only so the examine may show a
00000002.00000000: aec400c3       ! different value than written.

SROM> sb                          ! Set the base address to be used in
A> 2c0000000                      ! initializing clocks, timers and
00000002.c0000000                 ! memory refresh.
BaseAddr ON

SROM> dm                          ! Initialize the rest of the system.
A> 7600
D> 1

SROM> dm
A> 7680
D> bf00

SROM> dm
A> 7600
D> 6
```

```
SROM> dm
A> 7680
D> 0

SROM> dm
A> 7600
D> 5

SROM> dm
A> 7680
D> 400

SROM> dm
A> 2180
D> 74000000

SROM>dm
A> 2080
D> 1100

SROM> dm
A> 2080
D> 0
SROM> ba                          ! Disable use of base address.
00000002.c0000000
BaseAddr OFF

SROM> wa                          ! Use destination address as data
Wrt Addr ON                       ! for any write operations.

SROM> mt                          ! Perform memory test of entire 32MB.
A> 0
A> 2000000

SROM> bm                          ! Memory should contain its own
A> 100000                         ! address as data.
A> 100020
00000000.00100000: 00100000
00000000.00100004: 00100004
00000000.00100008: 00100008
00000000.0010000c: 0010000c
00000000.00100010: 00100010
00000000.00100014: 00100014
00000000.00100018: 00100018
00000000.0010001c: 0010001c

SROM>
```

2.7 Sample Session on the EB64+ and the AlphaPC 64

To run the mini-debugger on the EB64+ or the AlphaPC 64, set up the BIU_CTL register with the backup cache (Bcache) disabled. For example:

```
BIU_CTL :   0000004E 4001E664    ! for EB64+
            0000004E 4001E644    ! for AlphaPC 64.
```

After main memory is initialized, the Bcache can be enabled as shown in the EB64+ sample log. The example assumes a system with a 2MB Bcache. For a different Bcache size, change bits [30:28] in the BIU_CTL register (refer to *DECchip 21064 and DECchip 21064A Alpha AXP Microprocessors Hardware Reference Manual* for encoding) and the Tag Enable Register in the DECchip 21071 (see the *DECchip 21071 and DECchip 21072 Core Logic Chipsets Data Sheet* for encoding).

To initialize the memory controller for a 32-MB bank of memory in bank 0, the following values must be loaded into the Digital Semiconductor 21071-CA memory control registers:

Register	Data
Global Timing Register	00000025
Refresh Timing Register	00000444
Bank 0 Timing Register A	00002684
Bank 0 Timing Register B	00000C01
Bank 0 Base Address Register	00000000
Bank 0 Configuration Register	000000EB

Finally, all of memory must be written to initialize data parity.

To initialize and enable the Bcache, follow these steps:

1. Initialize the cache tag RAMs.

   This is done by enabling the cache in the Digital Semiconductor 21071-CA and configuring the Digital Semiconductor 21071-CA to ignore the tag, and to allocate blocks in the cache on write operations.

2. Perform a memory write to every address from address 0 up to the size of the cache.

   This will load good data, tag, and tag control parity into the Bcache.

3. Configure the Digital Semiconductor 21071-CA to use the Bcache tags, and set the BIU_CTL register in the Alpha 21064 to enable the Bcache.
4. Enable the Dcache.

Before accessing the PCI bus on the EB64+, several registers in the Digital Semiconductor 21071-DA must be initialized.

- The PCI Master Latency Timer must be set up before any PCI accesses are attempted.
- The address extension registers (HAXR*x*) that are used for different types of PCI accesses must also be initialized.

The EB64+ sample log file initializes HAXR2 before attempting a PCI configuration cycle.

2.7.1 EB64+ Sample Log File

The following sample log file initializes the EB64+. A log file for the AlphaPC 64 would be the same with the exception of the BIU_CTL register. See the comments in the log file.

```
   !  Sample EB64+ initialization Log file.
V00001c01
SROM> dc                        ! Turn BCache off by writing to BIU_CTL
IPR> bctl                       ! register. For AlphaPC 64 use 4E.4001.E644.
D> 4e4001e664
*BCTL 0000004e.4001e664

SROM> sb                        ! Set base for easier access to DC21071 regs.
A> 180000000
00000001.80000000
BaseAddr ON

SROM> dm                        ! Init Global Timing Register.
A> 200
D> 25

SROM> dm                        ! Init Refresh Timing Register.
A> 220
D> 444

SROM> dm                        ! Init Bank 0 Timing Register A.
A> c00
```

```
D> 2684

SROM> dm                        ! Init Bank 0 Timing Register B.
A> e00
D> c01

SROM> dm                        ! Set Bank 0 Base Address Reg to 0.
A> 800
D> 0

SROM> dm                        ! Init Bank 0 Configuration Register
A> a00                          ! to 32MB of memory.
D> eb


SROM> ba                        ! Disable use of base address.
00000001.80000000
BaseAddr OFF

SROM> fm                        ! Write good data and parity to all of
A> 0                            ! memory.
A> 2000000                      ! Total memory size = 32MB.
D> 0

! At this point the memory subsystem has been initialized and configured
! for a 32MB in bank 0 using 2Mbx36 SIMMS.  The BCache and DCache are off.

SROM> wa
Wrt Addr ON

SROM> mt                        ! Perform memory test.
A> 0
A> 2000000                      ! Total memory size = 32MB.

SROM> wa
Wrt Addr OFF

! Initialize the BCache and DCache.

SROM> ba
00000001.80000000
BaseAddr ON
SROM> dm                        ! Write General Control Register.
```

```
A> 0                          ! Enable Bcache, ignore tag, and allocate
D> 1b4                        ! on write operations.

SROM> dm                      ! Clear the Tag Enable Register.
A> 60
D> 0

SROM> ba
00000001.80000000
BaseAddr OFF

SROM> fm                      ! Write to entire cache to init cache
A> 0                          ! tags and data.
A> 200000                     ! Set this to BCache size = 2MB.
D> 0
SROM> ba
00000001.80000000
BaseAddr ON

SROM> dm                      ! Set Tag Enable Register to 2MB BCache.
A> 60
D> 3fe0

SROM> dm                      ! Clear the ignore tag bit previously set.
A> 0
D> b4

SROM> ba
00000001.80000000
BaseAddr OFF

SROM> dc                      ! Enable the BCache in the CPU.  For
IPR> bctl                     ! AlphaPC 64 use 4E.4001.E645.
D> 4e4001e665
*BCTL   0000004e.4001e665

! At this point the BCache is initialized and enabled.

SROM> dc                      ! Enable DCache and machine checks.
IPR> abox
D> 42a
*ABOX   00000000.0000042a
```

```
! The next section inits the PCI interface of the DC21071.

SROM> dm                        ! Set PCI Master Latency Timer to 255.
A> 1a00001e0
D> ff

SROM> dm                        ! Init HAXR2.
A> 1a00001c0
D> 0

SROM> em                        ! Perform a PCI configuration read of SIO.
A> 1e0080000
00000001.e0080000: 04848086


SROM> wa
Wrt Addr ON

SROM> mt
A> 0
A> 2000000

SROM>
```

2.8 Sample Session on the EB66 and EB66+

To run the mini-debugger on the EB66 or the EB66+, first determine the memory and Bcache size for your motherboard. The jumper settings and memory are described in the *DECchip 21066 and DECchip 21068 Alpha AXP Evaluation Board User's Guide* and the *Alpha 21066A Microprocessor Evaluation Board (EB66+) User's Guide*. The EB66 supports CPU speeds up to 166 MHz and 2 banks of memory. The EB66+ supports CPU speeds up to 233 MHz and 3 banks of memory.

To set up the mini-debugger for the EB66 or the EB66+, follow these steps:

1. Ensure that the "Write wrong ECC" bits are cleared in the error status register.

2. Set the Global Timing Register (GTR). For example, if you use 70-ns SIMMs, use the value from the following table, corresponding to your CPU speed:

CPU Speed	EB66 GTR Value	EB66+ GTR Value
66 MHz	0x007C10A1	0x007C10A2
100 MHz	0x10BC1CE3	0x00BC1CC3
166 MHz	0x11343185	0x01343146
200 MHz	0x21743DE7	0x01743DA6
233 MHz	0x21AC4629	0x11B445C8
266 MHz	0x31EC528A	0x11EC5249
300 MHz	0x322C5EEB	0x122C5EBA

3. Initialize the Bcache.

   Configure the memory banks for the fully supported memory size to avoid any illegal addresses that could cause the memory subsystem to fail to initialize all backup cache entries. When the system is powered on, the backup cache contains UNPREDICTABLE data in the tag RAMs. As the cache is swept for initialization, the old blocks (called the dirty victim blocks) are written back to main memory. These victim write operations occur based on the tags that store the upper part of the address location for the dirty blocks of memory. These UNPREDICTABLE tags cause victim write operations to UNPREDICTABLE addresses in memory without regard to the quantity of memory actually contained in the system.

4. Perform eight consecutive RAS cycles to each memory bank to “wake up” the DRAMs.

   This is done by reading (or writing) from each bank eight times. Space the read operations 32KB apart to avoid page mode cycles. In the EB66 sample log file shown in Section 2.8.1, this procedure is replaced by a fill-memory command to the entire memory, which has the same effect but requires less typing.

   Note that the caches are disabled at this point so that the read operations (and write operations) go directly to the DRAMs.

5. Load the cache control register (CAR) to set the Bcache timing and size as well as to disable ECC and parity.

   The following table displays the bank timing register (BTR) value for different CPU speeds for 70-ns SIMMs:

CPU Speed	EB66 BTR Value	EB66+ BTR Value
66 MHz	0x00202200	0x00022311
100 MHz	0x00423310	0x00023411
166 MHz	0x00645521	0x00046722
200 MHz	0x00866721	0x00067833
233 MHz	0x00C67832	0x00068933
266 MHz	0x00C88932	0x0128AB44
300 MHz	0x02EAAB53	0x012ABC55

6. Write all locations in Bcache.

   The following table displays the memory range for the cache size:

Cache Size	Memory Range
256KB	0 → 40000
512KB	0 → 80000
1MB	0 → 100000

7. Enable error checking in the Bcache.
8. Set the bank configuration registers (BCR*x*) and bank mask registers (BMR*x*) to match what is on the motherboard.

The following table displays the BCR and BMR values for the selected bank size.

Bank Size	SIMM Type	BCR	BMR
8MB	1MB×36	44C0	700000
16MB	2MB×36	64C0	F00000
32MB	4MB×36	45C0	1F00000
64MB	8MB×36	65C0	3F00000
128MB	16MB×36	47C0	7F00000

In the EB66 sample log file shown in Section 2.8.1, there are two banks with 4MB×36 SIMMs for a total memory size of 64MB.

9. Write good parity and ECC to memory by writing all memory locations. (memory size×1024×1024) –4. In the sample log file, this value is 3FFFFFC for 64MB of memory.

10. Initialize the error status register (ESR).

    The ESR may have powered on with some error bits set, or error bits may have been set while sweeping the Bcache. Write a 1 to all the write-1-to-clear bits to initialize this register.

11. By default, the PCI bus is disabled. To enable access to the PCI bus and configuration registers you must clear the PCI RST signal.

2.8.1 EB66 Sample Log File

The following sample log file initializes the EB66.

```
  !    Sample EB66 initialization Log File.
  !    Differences for EB66+ are noted in comment field.

V00001c01
SROM> sb                    ! Set base address for memory controller.
A> 120000000
00000001.20000000
BaseAddr ON

SROM> dm                    ! Ensure that the "Write wrong ECC" bits are
A> 68                       ! cleared in the Error Status Register.
D> 0
```

```
SROM> dm                    ! Set Global Timing Register (GTR) with
A> 60                       ! refresh enabled, bit [18] = 1.
D> 11343185                 ! Based on 70 ns SIMMs and 166 MHz CPU.
                            ! See table for other CPU speeds.

  ! Configure the memory banks for the fully supported memory size.

SROM> dm
A> 0                        !  BCR0.
D> 43c0                     !  Ignores ECC bit.  Bank Base 0x0.

SROM> dm
A> 8                        !  BCR1.
D> 80043c0                  !  Ignores ECC bit.  Bank Base 0x8000000.

SROM> dm
A> 10                       !  BCR2.
D> 100043c0                 !  Ignores ECC bit. Bank Base 0x10000000.

SROM> dm
A> 18                       !  BCR3.
D> 180043c0                 !  Ignores ECC bit. Bank Base 0x18000000.

SROM> dm
A> 20                       !  BMR0. Set to max size bank size = 128MB.
D> 7f00000
SROM> dm
A> 28                       !  BMR1. Set to max size bank size = 128MB.
D> 7f00000

SROM> dm
A> 30                       !  BMR2. Set to max size bank size = 128MB.
D> 7f00000

SROM> dm
A> 38                       !  BMR3. Set to max size bank size = 128MB.
D> 7f00000

SROM> dm
A> 40                       !  Bank Timing 0.
D> 645521                   !  Based on 70 ns SIMMs and 166 MHz CPU speed.
                            !  For EB66+ use 68933 for 233 MHz CPU speed.
```

```
SROM> dm
A> 48                          !  Bank Timing 1.
D> 645521                      !  For EB66+ use 68933.

SROM> dm
A> 50                          !  Bank Timing 2.
D> 645521                      !  For EB66+ use 68933.

SROM> dm
A> 58                          !  Bank Timing 3.
D> 645521                      !  For EB66+ use 68933.

SROM> ba
00000001.20000000
BaseAddr OFF

    ! Perform 8 consecutive RAS cycles to each memory bank to “wake up” the
    ! DRAMs.

SROM> fm
A> 0
A> 20000000                    ! Maximum memory size supported.
D> 0                           ! This may take several seconds.

   ! Load CAR to set Bcache timing, size, and disable ECC and parity.
   ! The exact BCache timing varies depending on CPU and cache RAM speeds.
   ! You can set the read and write speeds to a maximum value. For
   ! example, use 7f61 for a 512KB cache with maximum read and write speeds
   ! allowed.

SROM> dm
A> 120000078
D> 6b41

   ! Write enough locations to sweep the Bcache.
   ! Note that the Bcache is allocated on write operations.

SROM> fm
A> 0
A> 100000                      ! Size of BCache.
D> 0
```

```
SROM> ba
00000001.20000000
BaseAddr ON

SROM> dm                        ! Enable ECC checking (bit 4) and tag
A> 78                           ! parity (bit 2) in the Bcache.
D> 6b55                         !  Same value as before but with these
                                ! two bits set.

   ! Setting Bank Configuration Registers for 4MB x 36 SIMMs.
   ! Do this only for installed banks.

SROM> dm
A> 0
D> 45c0

SROM> dm
A> 8
D> 20045c0

SROM> dm
A> 10
D> 0

SROM> dm
A> 18
D> 0
   ! Setting Bank Mask Registers for 4MB x 36 SIMMs.

SROM> dm
A> 20
D> 1f00000

SROM> dm
A> 28
D> 1f00000

SROM> ba
00000001.20000000
BaseAddr OFF

   ! Write good parity/ECC to memory by writing all memory locations.
```

```
SROM> fm
A> 0
A> 4000000                    ! 64MB.
D> 0

  ! Initialize the Error Status Register.

SROM> ba
00000001.20000000
BaseAddr ON

SROM> em
A> 68
00000001.20000068: 00001003

SROM> dm
A> 68
D> 1003

SROM> em
A> 68
00000001.20000068: 00000000

  ! Clear PCI RST signal to allow the PCI bus to run.



SROM> dm
A> 1800000c0
D> 0
SROM> dm                  ! Enable access to the PCI configuration registers.
A> 180000020
D> 0

SROM>
```

2.9 Sample Session on the EB164

To run the mini-debugger as a standalone program on the EB164, the following setup needs to be performed. The mini-debugger initializes the CPU after baud detection (Section 2.5.1). You must also initialize the rest of the board.

To initialize the EB164, follow these steps:

1. Flush the secondary cache (L2) and turn on only one set. The reason to flush the secondary cache is to prevent any type of parity errors; only one set is turned on to facilitate read operations that need to go all the way out to memory and not be cached.
2. Turn off the Bcache to facilitate read operations that need to go all the way out to memory and not be cached. The Bcache configuration register must have the read and write speeds set to the ratio of the sysclock to CPU. Refer to the chapter on configuration jumpers in the board's user's guide to determine this ratio.
3. Initialize the CIA control register.
4. Initialize the CIA acknowledge register to disable the Bcache Victim Acknowledge signal.
5. Initialize the memory control register. This register controls the refresh rate, and memory width.
6. Set the bank timing registers. EB164 uses only one of these three registers.
7. Set MBA0, which controls the only memory bank in EB164. The value is dependent on the amount of memory and type of SIMMs used. Refer to the following table:

Memory Size	SIMM Type	MBA Value
32MB	1Mx36 (4MB)	10000011
64MB	2Mx36 (8MB)	10008011
128MB	4Mx36 (16MB)	10000073
256MB	8Mx36 (32MB)	10008073
512MB	16Mx36 (64MB)	100001F5

8. Wake up the memory by performing eight consecutive RAS cycles to each SIMM side.
9. Turn on Dcache, Bcache, and all three sets in the secondary cache.
10. Initialize memory and the caches by writing to memory.

11. At this point, memory initialization is complete. If you want to perform I/O tests, then you need to initialize that part of the system and:
    a. Reset the ISA bus.
    b. Configure SIO, enabling accesses to RTC, configuration RAM configuration jumpers), and flash ROM space.

2.9.1 EB164 Sample Log File

The following sample log file initializes the EB164.

```
   ! EB164 Log.


   ! Flush secondary cache, set block size to 64 bytes
   ! and turn on set S0 only to facilitate memory
   ! accesses.  Note that the secondary cache can't
   ! be turned off completely.


SROM> dm
A> fffff000a8
D> 3002
   ! Check previous write operation.

SROM> em
A> fffff000a8
000000ff.fff000a8: 00003002
   ! Note that the Bcache has been disabled and error reporting has
   ! been turned off.

SROM> dc
IPR> bctl
D> 8050
*BCTL   00000000.00008050
   ! The Read and Write speeds in the Bcache Cfg register must be set to
   ! the sysclock-to-cpu ratio.  This examples shows a ratio of 8.

SROM> dc
IPR> bcfg
D> 1e22880
*BCFG   00000000.01e22880
```

```
SROM> dm                    ! Set the CIA_CTRL register.
A> 8740000100
D> 2100c0f1

SROM> dm                    ! Set the PCI timer register.
A> 87400000C0
D> ff00
   ! Set the CIA_CACK_EN register.  Note that the
   ! Bcache victim bit has been set to 0 since the
   ! Bcache has been disabled.

SROM> dm
A> 8740000600
D> 0

SROM> sb                    ! Set base address for memory controller.
A> 8750000000
00000087.50000000
BaseAddr ON

SROM> dm                    ! Set the refresh rate, Bcache size to 0 (disabled)
A> 0                        ! and memory width to 256-bits.
D> 1fe01

SROM> dm                    ! Set Bank Timing Register 1.
A> b40
D> 60208140

   ! Set MBA0 which controls the only memory bank on
   ! EB164.  This example applies to all 8 SIMMs fully
   ! populated with 2Mbx36 SIMMs (8MBytes SIMMs) which
   ! gives a total of 64MBs.  See table for other values.

SROM> dm
A> 600
D> 10008011

SROM> ba
00000087.50000000
BaseAddr OFF

   ! Perform 8 consecutive RAS cycles to “wake up” memory.
```

```
SROM> fm
A> 0
> 4000000                  ! Total memory size.
D> 0                       ! This may take a few seconds.

SROM> dc                   ! Turn on the DCache.
IPR> dcmd
D> 1
*DCMD   00000000.00000001

SROM> dc                   ! Turn on the Bcache.
IPR> bctl
D> 8051
*BCTL   00000000.00008051

   ! The BCFG value shown is for a 266MHz CPU, with 2MB, 10ns Bcache.
   ! For other configurations, use maximum read and write speeds allowed.

SROM> dc
IPR> bcfg
D> 1e22772
*BCFG   00000000.01e22772

SROM> ec
BCtl    00000000.00008051
BCfg    00000000.01e22772
....                       ! The rest of the registers were omitted here.
DcMd    00000000.00000001
....                       ! The rest of the registers were omitted here.

SROM> dm                   ! Tell memory controller the size of Bcache (2MB).
A> 8750000000
D> 1fe21

SROM> dm                   ! Enable CACK of Bcache Victims.
A> 8740000600
D> 8

SROM> dm                   ! Enable all 3 sets in the secondary cache and
A> fffff000a8              ! leave at 64-byte blocks.
D> f000
```

```
SROM> fm                    ! Fill memory to obtain good ECC.
A> 0
A> 4000000                  ! Total memory size in board.
D> 12345678

SROM> bm
A> 0
A> 20
00000000.00000000: 12345678
00000000.00000004: 12345678
00000000.00000008: 12345678
00000000.0000000c: 12345678
00000000.00000010: 12345678
00000000.00000014: 12345678
00000000.00000018: 12345678
00000000.0000001c: 12345678

SROM> bm
A> 200000
A> 200020
00000000.00200000: 12345678
00000000.00200004: 12345678
00000000.00200008: 12345678
00000000.0020000c: 12345678
00000000.00200010: 12345678
00000000.00200014: 12345678
00000000.00200018: 12345678
00000000.0020001c: 12345678

SROM> bm
A> 400000
A> 400020
00000000.00400000: 12345678
00000000.00400004: 12345678
00000000.00400008: 12345678
00000000.0040000c: 12345678
00000000.00400010: 12345678
00000000.00400014: 12345678
00000000.00400018: 12345678
00000000.0040001c: 12345678
```

```
SROM> sb                    ! Set base address for PCI configuration
A> 8700000000               ! space.
00000087.00000000
BaseAddr ON

SROM> dm                    ! Reset the ISA bus.
A> 809a0                    ! Assert the reset signal.
D> 5800

SROM> dm                    ! Deassert the reset signal.
A> 809a0
D> 5000

SROM> dm                    ! Set UBCSA register to allow access to
A> 809c0                    ! RTC, keyboard, lbios and xbios.
D> c30000

SROM> dm                    ! Set UBCSB register to allow access to
A> 809e0                    ! configuration jumpers (address 0x801-2).
D> ff000000

SROM> em                    ! Read SIO identification registers.
A> 80000                    ! The value shown is the Intel ID for
00000087.00080000: 04848086         ! this part.

SROM> ba
00000087.00000000
BaseAddr OFF

SROM> dm                    ! Write out to the LEDs post-card.  You
A> 8580001000               ! should see an “F0” on it.
D> f0

SROM> em                    ! Read first byte of flash.  If it has an
A> 86fff80000               ! image with the standard ROM header, you
00000086.fff80000: 5a5ac3c3         ! will see this signature.
```

2.10 Sample Session on the AlphaPC 164

To run the mini-debugger as a standalone program on the AlphaPC 164, the following setup needs to be performed. The mini-debugger performs minimal initialization of the CPU and the user must initialize the rest of the board.

To initialize the AlphaPC 164, follow these steps:

1. Flush the secondary cache (L2) and turn on only one set. The reason to flush the secondary cache is to prevent any type of parity errors; only one set is turned on to facilitate read operations that need to go all the way out to memory and not be cached.

2. Turn off the Bcache to facilitate read operations that need to go all the way out to memory and not be cached. The Bcache configuration register must have the Read and Write speeds set to the ratio of the sysclock to CPU. Refer to the chapter on configuration jumpers in the board's user's guide to determine this ratio.

   Depending on the CPU speed, select the appropriate value for the BC_CONFIG and BC_CONTROL registers from the following table. If the Bcache is to be turned off, the read and write fields must be set to the cpu-to-sys clock ratio in BC_CONFIG and have the low bit cleared in BC_CONTROL.

Chip Speed	BC_CONFIG	BC_CONTROL
300 MHz	00000000001F21571	0000000444068051
333 MHz	00000000001F21581	00000004C4008051
366 MHz	00000000001F21691	00000004C4008051
400 MHz	00000000001F31691	00000004C4008051
433 MHz	00000000001F317A1	00000004C4008051
466 MHz	00000000003F318B1	0000000444068051
500 MHz	00000000003F418B1	0000000444068051

3. Initialize the CIA control register.
4. Initialize the CIA acknowledge register to disable the Bcache Victim Acknowledge signal.
5. Initialize the memory control register. This register controls the refresh rate and memory width.
6. Set the bank timing registers. AlphaPC 164 uses only one of these three registers.
7. Enter the appropriate values into the MBA registers, which control the only memory bank in AlphaPC 164. The value is dependent on the amount of memory and type of SIMMs used, as shown in the following table:

Note: Because AlphaPC 164 uses the alternate memory mode of the Digital Semiconductor 21172, programming the MBA registers is slightly different from the EB164. All 8 MBA registers must be programmed: the first 4 control side 0 of the SIMMs and the last 4 control side 1. As a result, note in the following table that the s1_valid bit in the MBA registers is not set when using double-sided SIMMs. The values for the last 4 registers are computed in the same manner as the first 4, and they only differ in the base address field ([25:16]).

SIMM Density	SIMM Size	Sides	128-bit Memsize	Mask	Row Type	MBA Value [15:00]	Wrap/Side 2 Addresses
1M×36	4MB	1	16MB	00000	10×10	0001	0×100000
2M×36	8MB	2	32MB	00000	10×10	0001	0×100000 /16MB
4M×36	16MB	1	64MB	00011	11×11	0033	0×200
8M×36	32MB	2	128MB	00011	11×11	0033	0×200/64MB
16M×36	64MB	1	256MB	01111	12×12	00F5	0×100
32M×36	128MB	2	512MB	01111	12×12	00F5	0×100/256MB

SIMM Density	SIMM Size	Sides	128-bit Memsize	Mask	Row Type	MBA Value [15:00]	Wrap/Side 2 Addresses
1M×36	4MB	1	32MB	00001	10×10	0011	0×100000
2M×36	8MB	2	64MB	00001	10×10	0011	0×100000 /16MB
4M×36	16MB	1	128MB	00111	11×11	0073	0×200
8M×36	32MB	2	256MB	00111	11×11	0073	0×200/64MB
16M×36	64MB	1	512MB	11111	12×12	01F5	0×100
32M×36	128MB	2	1GB	11111	12×12	01F5	0×100/256MB

8. Wake up the memory by performing eight consecutive RAS cycles to each SIMM side. This can also be done by writing to the entire memory range.
9. Turn on Dcache, Bcache, and all three sets in the secondary cache.
10. Initialize memory and the caches by writing to memory.
11. At this point, memory initialization is complete. If you want to perform I/O tests, then you need to initialize that part of the system and:
    a. Reset the ISA bus.
    b. Configure SIO, enabling accesses to RTC, configuration RAM (configuration jumpers), and flash ROM space.
    c. Initialize the RTC clock in the SMC chipset.

2.10.1 AlphaPC 164 Sample Log File

The following sample log file initializes the AlphaPC 164.

```
   ! AlphaPC 164 Log.
   !
   ! Flush secondary cache, set block size to 64 bytes
   ! and turn on set S0 only to facilitate memory
   ! memory accesses. Note that the secondary cache can't
   ! be turned off completely.

V00001c01
SROM> dm
A> fffff000a8
D> 3002

SROM> em                        ! Check previous write operation.
A> fffff000a8
000000ff.fff000a8: 00003002

SROM> dc                        ! Note that the Bcache has been disabled
IPR> bctl                       ! and error reporting has been turned off.
D> 8040
*BCTL   00000000.00008040

      ! The read and write speeds in the Bcache Cfg register must be set to
      ! the sysclock-to-cpu ratio. This example shows a ratio of 11 (0xB).

SROM> dc
IPR> cfg
D> 1f21bb0
*BCFG   00000000.01f21bb0
```

```
SROM> ec                    ! Note changes to BCtl and BCfg.
BCtl    00000000.00008040
BCfg    00000000.01f21bb0
Icsr    000000c2.4e000000
PalB    00000000.00000000
ExAd    00000000.00000000
Ipl     00000000.0000001e
Int     00000000.00000015
Isr     00000000.00200000
IcPE    00000000.00000000
DcMd    00000000.00000000
DcPE    00000000.00000000

SROM> dm                    ! Set the CIA_CTRL register.
A> 8740000100
D> 2104c0f5

SROM> dm                    ! Set the PCI timer register.
A> 87400000C0
D> ff00
      ! Set the CIA_CACK_EN register. Note that the
      ! Bcache victim bit has been set to 0 since the
      ! Bcache has been disabled.

SROM> dm
A> 8740000600
D> 0

SROM> sb                    ! Set base address for memory controller.
A> 8750000000
00000087.50000000
BaseAddr ON

  ! Set the refresh rate, Bcache size to 0 (disabled) and memory
  ! width to 256-bits (all 8 SIMM slots populated).  If 128-bit
  ! width is being used, clear the low bit of this register.

SROM> dm
A> 0                        ! Memory configuration register.
D> 2001fe01
```

```
SROM> dm                        ! Set Bank Timing Register #1.
A> b40
D> 60208142

   ! In AlphaPC 164, all 8 memory address registers (MBAs) must
   ! be programmed.  The first 4 cover side 0 of the
   ! SIMMs and the last 4, side 1 (if present).  In this
   ! example all 8 SIMMs fully populated with 2Mbx36 SIMMs
   ! (8MBytes SIMMs) which gives a total of 64MBs.  The
   ! first 4 MBA registers cover side 0, whose base address
   ! (bits 25:16) is 0.  Side 1's base address is half
   ! the total memory size (32MB = 0x200000) and hence
   ! bits 25:16 are programmed with 0x2.

SROM> dm
A> 600                          ! MBA0.
D> 11

SROM> dm
A> 680                          ! MBA2.
D> 11

SROM> dm
A> 700                          ! MBA4.
D> 11

SROM> dm
A> 780                          ! MBA6.
D> 11

SROM> dm
A> 800                          ! MBA8.
D> 20011

SROM> dm
A> 880                          ! MBAA.
D> 20011

SROM> dm
A> 900                          ! MBAC.
D> 20011
```

```
SROM> dm
A> 980                          ! MBAE.
D> 20011

SROM> ba
00000087.50000000
BaseAddr OFF

       ! Perform 8 consecutive RAS cycles to “wake up” memory.

SROM> fm
A> 0
A> 4000000                      ! Total memory size.
D> 0                            ! This may take a few seconds.

SROM> dc                        ! Turn on the DCache.
IPR> dcmd
D> 1
*DCMD   00000000.00000001

       ! The BCFG and BCTL values shown are for a 366MHz CPU, with
       ! 1MB Bcache.  For other configurations, see table.

SROM> dc
IPR> bcfg
D> 1f21691                      ! Sets the BCache timing.
*BCFG   00000000.01f21691

SROM> dc
IPR> bctl
D> 4c4008051                    ! Set the BCache control.
*BCTL   00000004.c4008051
```

```
SROM> ec                    ! Make sure changes are reflected here.
BCtl    00000004.c4008051
BCfg    00000000.01f21691
Icsr    000000c2.4e000000
PalB    00000000.00000000
ExAd    284c8800.00080000
Ipl     00000000.0000000e
Int     00000000.00000015
Isr     00000000.00200000
IcPE    00000000.00000000
DcMd    00000000.00000001
DcPE    00000000.00000000

SROM> dm                    ! Enable CACK of Bcache Victims.
A> 8740000600
D> 8

SROM> dm                    ! Enable all 3 sets in the secondary cache
A> fffff000a8               ! and leave at 64-byte blocks.
D> f000

SROM> fm                    ! Fill memory to obtain good ECC.
A> 0
A> 4000000                  ! Total memory size in board.
D> 12345678

      ! Clear any possible errors in the CPU and memory controller.

SROM> em
A> fffff000e8               ! SC_STAT register.
000000ff.fff000e8: 0001a86b

SROM> em
A> fffff00168               ! EI_STAT register.
000000ff.fff00168: f1ffffff

SROM> em
A> 8740008200               ! CIA_ERROR register.
00000087.40008200: 00000000

SROM> wa
Wrt Addr ON
```

```
SROM> mt                    ! Perform a memory test.
A> 0
A> 4000000

SROM> wa
Wrt Addr OFF

SROM> sb                    ! Set base address for PCI configuration
A> 8700000000               ! space.
00000087.00000000
BaseAddr ON

SROM> dm                    ! Reset the ISA bus.
A> 809a0                    ! Assert the reset signal.
D> 4800

SROM> dm                    ! Deassert the reset signal.
A> 809a0
D> 4000

SROM> dm                    ! Set UBCSA register to allow access to
A> 809c0                    ! lbios and xbios.
D> c00000

SROM> dm                    ! Set UBCSB register to allow access to
A> 809e0                    ! configuration jumpers (address 0x801-2)
D> ff000000

SROM> em                    ! Read SIO identification registers.
A> 80000                    ! The value shown is the Intel ID for
00000087.00080000:04848086  ! this part.

SROM> ba
00000087.00000000
BaseAddr OFF

SROM> dm                    ! Write out to the LEDs post-card. You
A> 8580001000               ! should see an “F0” on it.
D> f0
```

```
SROM> em                    ! Read first byte of flash. If it has an
A> 86fff80000               ! image with the standard ROM header, you
00000086.fff80000:5a5ac3c3  ! will see this signature.
        ! Enable the Real Time Clock in the SMC chipset.

SROM> dm                    ! Put SMC in configuration mode
A> 8580006e00               ! by writing to I/O address 0x370
D> 55                       ! the value 0x55 twice.

SROM> dm
A> 8580006e00
D> 55

SROM> dm
A> 8580006e00
D> 20                       ! Read the device ID and rev.

SROM> em
A> 8580006e20               ! Data port for SMC is at I/O address 0x371.
00000085.80006e20:00000200  ! It should contain this value.

SROM> dm                    ! Set device number to RTC.
A> 8580006e00               ! All the configuration commands
D> 7                        ! that follow will apply to RTC only.

SROM> dm
A> 8580006e20
D> 0600                     ! RTC.

SROM> dm
A> 8580006e00
D> 70                       ! Select the interrupt register.

SROM> dm
A> 8580006e20
D> 0800                     ! Configure to interrupt as IRQ8.

SROM> dm
A> 8580006e00
D> 30                       ! Activate the RTC.
```

```
SROM> dm
A> 8580006e20
D> 0100

SROM> dm                    ! Put SMC in run mode by writing
A> 8580006e00               ! 0xAA to address 0x370.
D> AA                       ! The RTC is now ready to be used.
```

2.11 Sample Session on the AlphaPC 164LX

To run the mini-debugger as a standalone program on the AlphaPC 164LX, the following setup needs to be performed. The mini-debugger performs minimal initialization of the CPU and the user must initialize the rest of the board.

To initialize the AlphaPC 164LX, follow these steps:

1. Flush the secondary cache (L2) and turn on only one set. The secondary cache is flushed to prevent any type of parity errors; only one set is turned on to facilitate read operations that need to go all the way out to memory and not be cached.
2. Turn off the Bcache to facilitate read operations that need to go all the way out to memory and not be cached. The Bcache configuration register must have the read and write speeds set relative to the CPU speed.

   Depending on the CPU speed, select the appropriate value for the BC_CONFIG and BC_CONTROL registers from the following table. If the Bcache is to be turned off, clear the low bit in BC_CONTROL.

Chip Speed	BC_CONFIG	BC_CONTROL
466 MHz	00000000003F346A2	00000000C4008051
533 MHz	00000000007F457C2	00000000C6008051
600 MHz	00000000007F457D2	00000000C6008051

3. Initialize the general registers.
4. Initialize the memory bank registers.

BBARQ-1 registers receive the base address bits [33:24] in bits [15:6]. BCR0-1 registers receive the bank size as in the following table:

Bank Size	Register Value
16MB	2C
32MB	2A
64MB	28
128MB	26
256MB	24
512MB	22

Each BCR register is first written with the appropriate value from the preceding table and then again with the low bit set to enable the banks.

5. Wake up memory by performing eight consecutive RAS cycles to each DIMM side. This can also be done by writing the full memory range.
6. Turn on the Dcache, Bcache, and the secondary cache.
7. Initialize memory and the caches by writing to memory.
8. At this point, memory initialization is complete. If you want to perform I/O tests, then you need to initialize that part of the system and:
   a. Reset the ISA bus.
   b. Configure SIO, enabling accesses to RTC, configuration RAM (configuration jumpers), and flash ROM space.
   c. Initialize the RTC clock in the SMC chipset.

2.11.1 AlphaPC 164LX Sample Log File

The following sample log file initializes the AlphaPC 164LX.

```
! AlphaPC 164LX Log.
!
! Flush Scache, set block size to 64 bytes
! and turn on set S0 only to facilitate memory
! accesses. Note that the Scache can't
! be turned off completely.
```

```
V00001f01
SROM> dm                    ! SC_CTL register.
A> fffff000a8
D> 3002

SROM> em                    ! Check previous write operation.
A> fffff000a8
000000ff.fff000a8:00003002

SROM> dc                    ! Disable the Bcache and turn off
IPR> bctl                   ! error reporting.
D> 8040
*BCTL   00000000.00008040

      ! The read and write speeds in the Bcache Cfg register are set
      ! using the table based on the CPU speed. This example is for 466 MHz.

SROM> dc
IPR> bcfg
D> 3f346a2
*BCFG   00000000.03f346a2

SROM> dm                    ! Set the control register.
A> 8740000100
D> f1

SROM> dm                    ! Set the flash control register.
A> 8740000200
D> 0

SROM> dm                    ! Set CCR (clock control register).
A> 8780000000
D> 029040631

SROM> dm                    ! Set GTR (global timing register).
A> 8750000200
D> 332

SROM> dm                    ! Set RTR (refresh timing register).
A> 8750000300
D> 750
```

```
SROM> dm                    ! Set MCR (memory control register).
A> 8750000000
D> 3a1401

SROM> dm                    ! Set RTR again to force a
A> 8750000300               ! refresh after the MCR has been set.
D> 750

   ! Set the memory bank registers for two banks of DIMMs totalling
   ! 128MB.

SROM> dm                    ! Set memory BBAR0
A> 8750000600               ! (base address register 0).
D> 0

SROM> dm                    ! Set memory BBAR1.
A> 8750000640
D> 100

SROM> dm                    ! Set memory BTR0
A> 8750000a00               ! (base timing register 0).
D> 22

SROM> dm                    ! Set memory BTR1.
A> 8750000a40
D> 22

SROM> dm                    ! Set memory BCR0
A> 8750000800               ! (base configuration register 0).
D> 28                       ! Write once with the bank disabled.

SROM> dm                    ! Set memory BCR1.
A> 8750000840
D> 28

SROM> dm                    ! Set memory BCR0.
A> 8750000800               ! Write a second time enabling the banks.
D> 29

SROM> dm                    ! Set memory BCR1.
A> 8750000840
D> 29
```

```
SROM> fm                    ! Fill 128MB with zeros.
A> 0
A> 8000000
D> 0
SROM> dc                    ! Turn on the DCache.
IPR> dcmd
D> 1
*DCMD   00000000.00000001

       ! Set BCFG and BCTL for 466 MHz CPU with 2MB Bcache.

SROM> dc                    ! Set CPU BC_CONFIG register.
IPR> bcfg
D> 3f346a2
*BCFG   00000000.03f346a2

SROM> dc                    ! Set CPU BC_CONTROL register.
IPR> bctl
D> c4008051
*BCTL   00000000.c4008051

SROM> ec                    ! Make sure changes are reflected here.
BCtl    00000000.c4008051
BCfg    00000000.03f346a2
Icsr    000000c2.4e000000
PalB    00000000.00000000
ExAd    00000481.05b4a80d
Ipl     00000000.0000001f
Int     00000000.0000001f
Isr     00000001.38200000
IcPE    00000000.00000000
DcMd    00000000.00000001
DcPE    00000000.00000000

SROM> dm                    ! Enable all 3 sets in the secondary cache
A> fffff000a8               ! and leave at 64-byte blocks.
D> f000

SROM> fm                    ! Fill memory to obtain good ECC.
A> 0
A> 8000000
D> 12345678
```

```
      ! Clear any possible errors in the CPU and memory controller.

SROM> em                    ! SC_STAT register.
A> fffff000e8
000000ff.fff000e8: 00000000

SROM> em                    ! EI_STAT register.
A> fffff00168
000000ff.fff00168: e5ffffff

SROM> dm                    ! PYXIS_ERR.
A> 8740008200               ! Writing 1 clears any errors.
D> 1

SROM> wa
Wrt Addr ON

SROM> mt
A> 0
A> 8000000

SROM> wa
Wrt Addr OFF

SROM> sb                    ! Set base address for PCI configuration
A> 8700000000               ! space.
00000087.00000000
BaseAddr ON

SROM> dm                    ! Reset the ISA bus.
A> 809a0                    ! Assert the reset signal.
D> 4800

SROM> dm                    ! Deassert the reset signal.
A> 809a0
D> 4000

SROM> dm                    ! Set UBCSA register to allow access
A> 809c0                    ! to lbios and xbios.
D> c0000
```

```
SROM> dm                    ! Set UBCSB register to allow access
A> 809e0                    ! to configuration jumpers
D> ff000000                 ! (address 0x801-2).

SROM> em                    ! Read SIO identification registers.
A> 80000                    ! The value shown is the Intel ID for
00000087.00080000:04848086  ! this part.

SROM> ba
00000087.00000000
BaseAddr OFF

SROM> dm                    ! Write out to the LEDs post-card. You
A> 8580001000               ! should see an “F0” on it.
D> f0

SROM> em                    ! Read first byte of flash. If it has an
A> 86fff80000               ! image with the standard ROM header, you
00000086.fff80000:5a5ac3c3  ! will see this signature.

        ! Enable the real-time clock in the SMC chipset.

SROM> dm                    ! Put SMC in configuration mode
A> 8580006e00               ! by writing to I/O address 0x370
D> 55                       ! the value 0x55 twice.

SROM> dm
A> 8580006e00
D> 55

SROM> dm
A> 8580006e00
D> 20                       ! Read the device ID and rev.

SROM> em
A> 8580006e20               ! Data port for SMC is at I/O address 0x371.
00000085.80006e20:  a5a5023c  ! It should contain this value.
                  !xxxx02xx     Only one byte is significant.

SROM> dm                    ! Set device number to RTC.
A> 8580006e00               ! All the configuration commands
D> 7                        ! that follow will apply to RTC only.
```

```
SROM> dm                    ! RTC.
A> 8580006e20
D> 0600

SROM> dm                    ! Select the interrupt register.
A> 8580006e00
D> 70

SROM> dm                    ! Configure to interrupt as IRQ8.
A> 8580006e20
D> 0800

SROM> dm                    ! Activate the RTC.
A> 8580006e00
D> 30

SROM> dm
A> 8580006e20
D> 0100

SROM> dm                    ! Put SMC in run mode by writing
A> 8580006e00               ! 0xAA to address 0x370.
D> AA                       ! The RTC is now ready to be used.
```

2.12 Onboard Machine Check Handler

The onboard machine check handler is useful in debugging certain memory faults. You must set bit [1] of the ABOX_CTL register to enable machine checks. For 21164-based boards, machine checks are always enabled. When a machine check is encountered, as it might be in a read, the machine check handler prints the following message followed by the SROM> prompt:

```
MCHK
Abox    00000000.00000002
Icsr    00000000.00ff0000
PalB    00000000.00000000
ExAd    00000000.00200cb4
DcSt    00000000.0000000b
Hirr    00000000.00000000
Hier    00000000.00001890
BCtl    000007f8.00000000
BiSt    00000000.000010c1
BiAd    00000001.e0000018
```

```
Syn     00000000.00000000
FiAd    00000000.00200950

SROM>
```

3
SROM Mini-Debugger Command Set

3.1 Overview

This chapter describes the Alpha Microprocessors SROM Mini-Debugger command set.

3.2 Command and User Interface Features

The following list describes some of the features of the mini-debugger command language:

- Uppercase or lowercase characters can be used interchangeably.
- Only the first two characters of a command line are significant; the rest are ignored.
- Numbers are input and output in hexadecimal format.
- Commands can be aborted at any time by pressing <Ctrl/C> (except for the xm command).
- For commands that prompt for input, pressing Return on an empty line defaults to a value of 0.
- In qw data mode, addresses are aligned to a quadword boundary (the three least significant bits of the addresses are zero). In all other modes, addresses are aligned to a longword boundary (the two least significant bits of the addresses are zero).
- In qw data mode, all reads and stores are performed using ldq/p and stq/p instructions, respectively. In all other modes, reads and stores use ldl/p and stl/p instructions, respectively.

- All stores are followed with two MB instructions. This keeps writes ordered (from the system's point of view) and prevents merging in the write queue. For more information on the MB instruction, see the CPU's hardware reference manual.
- For commands that use an address range, the ending address is not included in the range. The last read or store is performed in the immediately preceding quadword or longword, depending on the state of the qw data flag.

3.3 Command Summary

Table 3–1 summarizes the command set for the Alpha Microprocessors SROM Mini-Debugger. These commands are described in the following sections.

Table 3–1 Command Summary *(Sheet 1 of 2)*

Command	Description
	Flag Commands
!d	Enable/Disable negation of data to be written.
ba	Enable/Disable use of base address.
di	Enable/Disable display to screen.
fr	Enable/Disable follow-with-read flag.
fw	Enable/Disable follow-with-write flag.
lo	Enable/Disable command repetition.
qw	Enable/Disable quadword mode operations.
wa	Enable/Disable write address mode.
	Memory Commands
bm	Read block of memory addresses.
cm	Compare two sections of memory.
cp	Copy a block of memory.
fm	Fill memory range with data pattern.
mt	Peform simple memory test.

Table 3–1 Command Summary *(Sheet 2 of 2)*

Command	Description
	Deposit/Examine Commands
dc	Deposit internal CPU register.
dm	Deposit data to memory location.
ec	Examine internal CPU registers.
em	Examine data in a memory location.
	Print Commands
fl	Print current state of flags.
pr	Print contents of general purpose cpu registers.
	Load/Execute Commands
rt	Exit minidebugger.
st	Start executing at specified address.
xb	Begin execution of the last image loaded.
xm	Load external image to memory.
	Miscellaneous Command
sb	Set base address.

3.4 User Commands

This section contains a complete description and examples of the SROM Mini-Debugger commands. The commands are listed in alphabetical order. The Control Flags section in each command description lists the flags that affect the behavior of the command if the flag is enabled.

!d

The negate data (!d) command enables or disables the use of the one's complement of the data to be written by toggling the negate data flag.

Control Flags

Not applicable.

Description

When the negate data flag is enabled, writes will use the complement (negation) of the data specified by the user or automatically generated by the command in use.

The default state is off.

Example

In the following example, with the negate flag enabled, the data supplied by the user (0) is complemented and written to address 500000.

```
SROM> !d
Neg Data ON

SROM> dm
A> 500000
D> 0

SROM> em
A> 500000
00000000.00500000: ffffffff
```

With the fill memory command, the data written with each write is also complemented, resulting in every other write having the original data.

```
SROM> fm
A> 600000
A> 600020
D> ffff0000
```

```
SROM> bm
A> 600000
A> 600020
00000000.00600000: 0000ffff
00000000.00600004: ffff0000
00000000.00600008: 0000ffff
00000000.0060000c: ffff0000
00000000.00600010: 0000ffff
00000000.00600014: ffff0000
00000000.00600018: 0000ffff
00000000.0060001c: ffff0000
```

If the write address flag is also enabled, then data written is the complement of the destination address.

```
SROM> wa
Wrt Addr ON

SROM> fm
A> 500000
A> 500010

SROM> bm
A> 500000
A> 500010
00000000.00500000: ffafffff   ! This is the complement of x500000
00000000.00500004: ffaffffb
00000000.00500008: ffaffff7
00000000.0050000c: ffaffff3
```

ba

The base address flag (ba) command enables or disables the use of the base address set with the set base (sb) command. It does this by toggling the base address flag.

Control Flags

Not applicable.

Description

When the base address flag is enabled, the address entered with the sb command is added to addresses entered in any subsequent examine or deposit command.

The default state is off.

Example

To access addresses in the following range, 3.FFF80000 to 3.FFFFFFFF, it may be more convenient to enter its base address and work with offsets, rather than typing the absolute address every time.

```
SROM> sb
A> 3fff80000
00000003.fff80000
BaseAddr ON

SROM> bm
A> 0
A> 10
00000003.fff80000: 5a5ac3c3
00000003.fff80004: a5a53c3c
00000003.fff80008: 00000038
00000003.fff8000c: 00006579

SROM> ba
00000003.fff80000
BaseAddr OFF

SROM> em
A> 400000
00000000.00400000: 00400000
```

```
SROM> ba
0000003.fff80000
BaseAddr ON

SROM> bm
A> 30
A> 40
00000003.fff80030: 00000001
00000003.fff80034: 0000be8a
00000003.fff80038: 47ff041f
00000003.fff8003c: 47ff041f
```

bm

The block memory (bm) command displays the data read from a specified range of addresses.

Control Flags

ba (base address flag)
di (display flag)
fr (follow-with-read flag)
fw (follow-with-write flag)
lo (loop flag)
qw (quadword data flag)

Description

The block memory command reads data from a block of memory locations and prints it out to the screen if the display flag is enabled. The range is specified by first entering the starting address, followed by the ending address.

Example

The following example shows a block memory command display:

```
SROM> wa                ! Data for fill memory command
Wrt Addr ON

SROM> qw                ! Use quadword writes and 64-bit data.
QW ON

SROM> fm                ! Fill range with its own address
A> 400000
A> 400010


SROM> bm                ! Display range.
A> 400000
A> 400010
00000000.00400000: 00000000.00400000
00000000.00400008: 00000000.00400008
```

cm

The compare (cm) command compares two sections of memory and displays any differences.

Control Flags

ba (base address flag)
lo (loop flag)
qw (quadword data flag)

Description

The compare command checks the equality of two blocks of memory and prints any differences. The first two addresses specify the starting and ending addresses of the first block to be checked and the third input provides the starting address of the second block.

Example

In the following example, two blocks are filled with the same data and then two locations are changed. The compare command shows those two locations as having different data, even though the display flag is disabled.

```
SROM> fm                  ! Fill the first block with a known pattern.
A> 500000
A> 500020
D> 12345678

SROM> fm                  ! Fill the second block with a known pattern.
A> 600000
A> 600020
D> 12345678

SROM> dm                  ! Change first location.
A> 600004
D> bad

SROM> dm                  ! Change second location.
A> 60000c
D> deed
```

```
SROM> di                ! Does not affect the compare command.
Disp OFF

SROM> cm                ! Compare both sections.
A> 500000
A> 500020
A> 600000
00000000.00500004: 12345678
00000000.00600004: 00000bad
00000000.0050000c: 12345678
00000000.0060000c: 0000deed
```

cp

The copy (cp) command reads data from a range of addresses and writes it to another.

Control Flags

ba (base address flag)
lo (loop flag)
qw (quadword data flag)

Description

The copy command moves sections of data from one place in memory to another. The first two addresses specify the starting and ending addresses of the block to be moved. The third input provides the destination address for the copy.

Example

In the following example, a block starting at 400000 is filled with its own addresses and then copied to 500000:

```
SROM> wa                  ! Use the address as the data for the writes.
Wrt Addr ON

SROM> fm                  ! Fill this block with its own addresses.
A> 400000
A> 400020

SROM> cp                  ! Copy the block to 500000
A> 400000
A> 400020
A> 500000

SROM> bm                  ! This shows the data was moved to 500000
A> 500000
A> 500010
00000000.00500000: 00400000
00000000.00500004: 00400004
00000000.00500008: 00400008
00000000.0050000c: 0040000c
```

dc

The deposit CPU register (dc) command changes the contents of internal CPU registers.

Control Flags

Not applicable.

Description

The deposit CPU register command changes the contents of internal CPU registers. These registers are CPU dependent, so command input is different for each Alpha CPU. The following tables show the names assigned by the mini-debugger to the CPU registers; these are the only valid names that may be entered at the IPR prompt.

Table 3–2 CPU Register Names for the dc Command *(Sheet 1 of 2)*

Mini-debugger Name	CPU Register Name	Description
	21164	
BCtl*	BC_CONTROL	Bcache control
BCfg*	BC_CONFIG	Bcache configuration
Icsr	ICSR	Ibox control and status
PalB	PAL_BASE	PAL base address
DcMd	DC_MODE	Dcache mode
Ipl	IPLR	Interrupt priority level
	21064	
BCtl*	BC_CONTROL	Bcache control
Abox*	ABOX_CTL	Abox control
Icsr	ICSR	Ibox control and status
PalB	PAL_BASE	PAL base address

Table 3–2 CPU Register Names for the dc Command *(Sheet 2 of 2)*

Mini-debugger Name	CPU Register Name	Description
	21066	
Abox*	ABOX_CTL	Abox control
Icsr	ICSR	Ibox control and status
PalB	PAL_BASE	PAL base address

* Write-only registers whose values are obtained from a copy placed in PALtemp registers 1and 2 when they were written with the dc command. Therefore, changing these internal CPU registers overwrites the saved contents of the general-purpose CPU registers 1 and 2 (also stored in PALtemp 1 and 2), affecting the output of the pr command.

Example

In the following example, output on a DS21064 CPU is displayed:

```
SROM> dc                  ! Enable machine checks and CRD interrupts
IPR> abox
D> 6
*ABOX   00000000.00000006

SROM> ec
Abox    00000000.00000006
Icsr    00000000.00ff0000
PalB    00000000.00000000
ExAd    00000000.00200cac
DcSt    00000000.00000003
Hirr    00000000.00000000
Hier    00000000.00001890
BCtl    000007f8.00000000
BiSt    00000000.00001040
BiAd    00000000.01ffd010
Syn     00000000.00000000
FiAd    00000000.01ffd014
```

di

The display (di) command enables or disables the display of data to the screen by toggling the display flag.

Control Flags

Not applicable.

Description

When the display flag is enabled, the examine commands print the data obtained. When it is disabled, the read operations still take place but the data is not displayed. The cm and mt commands, which produce read operations of their own, ignore the state of this flag and always display the data if there is a mismatch.

The default state is on.

Example

In the following example, the em command performs a read operation and displays the data read in because the display flag is enabled. However, when the flag is disabled, the read operations performed by the bm command are not echoed to the screen.

```
SROM> em
A> 500000
00000000.00500000: ffffffff

SROM> di
Disp OFF

SROM> bm
A> 500000
A> 500010
```

The next example shows that the cm command is not affected by this flag's state and prints out data whenever a mismatch occurs.

```
SROM> di
Disp OFF

SROM> cm
A> 500000
A> 500010
A> 600000
00000000.00500000: ffffffff
00000000.00600000: 0000ffff
00000000.00500004: 000f0000
00000000.00600004: ffff0000
00000000.00500008: 000f0000
00000000.00600008: 0000ffff
00000000.0050000c: 000f0000
00000000.0060000c: ffff0000
```

dm

The deposit memory (dm) command writes a data pattern to one memory location.

Control Flags

!d (negate data flag)
ba (base address flag)
fr (follow-with-read flag)
fw (follow-with-write flag)
lo (loop flag)
qw (quadword data flag)
wa (write address flag)

Description

The deposit memory command writes a data pattern to the specified memory location. If the quadword flag is enabled, then 64 bits of data are written; otherwise, only 32 bits of data are used. The data pattern is provided by the user but can be affected by the state of the write address flag and the negate data flag.

Example

In this example, a continuous write to address 500000 is performed with the loop flag enabled. Note that because the quadword flag is disabled, only the low 32 bits of the data pattern entered are written.

```
SROM> di                ! Enable display mode
Disp ON

SROM> qw                ! Writes are performed 32-bits at a time.
QW OFF

SROM> lo                ! Repeat operation until a key is pressed.
Loop ON

SROM> dm                ! Only the low 32-bits are used in these writes.
A> 500000
D> 123456789abcdef
                        ! (key pressed)
```

```
SROM> em                ! Examine contents of address 500000
A> 500000
00000000.00500000: 89abcdef
00000000.00500000: 89abcdef
00000000.00500000: 89abcdef
                        ! (key pressed)
```

ec

The examine CPU registers (ec) command prints the contents of internal CPU registers.

Control Flags

Not applicable.

Description

The examine CPU registers command displays the contents of internal CPU registers. These registers are CPU dependent, so command output is different for each Alpha CPU. The following tables show the names assigned by the mini-debugger to the CPU registers:

Table 3–3 CPU Register Names for the ec Command *(Sheet 1 of 2)*

Mini-debugger Name	CPU Register Name	Description
	DS21164	
BCtl*	BC_CONTROL	Bcache control
BCfg*	BC_CONFIG	Bcache configuration
Icsr	ICSR	Ibox control and status
PalB	PAL_BASE	PAL base address
ExAd	EXC_ADDR	Exception address
Ipl	IPLR	Interrupt priority level
Int	INTID	Interrupt ID
Isr	ISR	Interrupt summary
IcPE	ICPERR_STAT	Icache parity error status
DcMd	DC_MODE	Dcache Mode
DcPE	DC_PERR_STAT	Dcache parity error status

Table 3–3 CPU Register Names for the ec Command *(Sheet 2 of 2)*

Mini-debugger Name	CPU Register Name	Description
	DS21064	
Abox*	ABOX_CTL	Abox control
Icsr	ICSR	Ibox control and status
PalB	PAL_BASE	PAL base address
ExAd	EXC_ADDR	Exception address
DcSt	DC_STAT	Data cache status
Hirr	HIRR	Hardware interrupt request
Hier	HIER	Hardware interrupt enable
BCtl*	BC_CONTROL	Bcache control
BiSt	BIU_STAT	Bus interface unit status
BiAd	BIU_ADDR	Bus interface unit address
Syn	FILL_SYNDROME	Fill syndrome
FiAd	FILL_ADDR	Fill Address
	DS21066	
Abox*	ABOX_CTL	Abox control
Icsr	ICSR	Ibox control and status
PalB	PAL_BASE	PAL base address
ExAd	EXC_ADDR	Exception address
DcSt	DC_STAT	Data cache status
Hirr	HIRR	Hardware interrupt request
Hier	HIER	Hardware interrupt enable

* Write-only registers whose values are obtained from a copy placed in PALtemp registers 1 and 2 when they were written with the dc command. Therefore, changing these internal CPU registers overwrites the saved contents of the general-purpose CPU registers 1 and 2 (also stored in PALtemp 1 and 2), affecting the output of the pr command.

Example

The following example shows the output of an ec command on a DS21064 CPU:

```
SROM> ec
Abox    00000000.00000002
Icsr    00000000.00ff0000
PalB    00000000.00000000
ExAd    00000000.00200cb4
DcSt    00000000.0000000b
Hirr    00000000.00000000
Hier    00000000.00001890
BCtl    000007f8.00000000
BiSt    00000000.000010c1
BiAd    00000001.e0000018
Syn     00000000.00000000
FiAd    00000000.00200950
```

em

The examine memory (em) command reads data from one memory location.

Control Flags

!d (negate data flag)
ba (base address flag)
di (display flag)
fr (follow-with-read flag)
fw (follow-with-write flag)
lo (loop flag)
qw (quadword data flag)

Description

The examine memory command reads data from the specified memory location and displays it on the screen if the display (di) flag is enabled. Depending on the state of the quadword data flag, 32 or 64 bits will be displayed.

Example

The following example shows output from an em command:

```
SROM> di                  ! Enable display mode
Disp ON

SROM> qw                  ! Reads are performed 64-bits at a time.
QW ON

SROM> lo                  ! Repeat operation until a key is pressed.
Loop ON

SROM> em                  ! Repeat read until a key is pressed.
A> 3fff80000
00000003.fff80000: a5a53c3c.5a5ac3c3
00000003.fff80000: a5a53c3c.5a5ac3c3
00000003.fff80000: a5a53c3c.5a5ac3c3
00000003.fff80000: a5a53c3c.5a5ac3c3
00000003.fff80000: a5a53c3c.5a5ac3c3
                          ! (key pressed)
```

fl

The flags (fl) command displays the current state of all flags.

Control Flags

Not applicable.

Description

The behavior of many of the mini-debugger commands can be affected by the state of one or more of eight available flags. The flags command displays on the screen the current state of these flags.

Example

In the following example, the default state of all flags, except the display flag, is off (disabled). Their states can be changed by issuing the appropriate command.

```
SROM> fl
FollowWr OFF
FollowRd OFF
BaseAddr OFF
Neg Data OFF
Wrt Addr OFF
Loop OFF
Disp ON
QW OFF

SROM> wa
Wrt Addr ON

SROM> !d
Neg Data ON
```

```
SROM> fl
FollowWr OFF
FollowRd OFF
BaseAddr OFF
Neg Data ON
Wrt Addr ON
Loop OFF
Disp ON
QW OFF
```

fm

The fill memory (fm) command writes data to the specified range of addresses.

Control Flags

!d (negate data flag)
ba (base address flag)
fr (follow-with-read flag)
fw (follow-with-write flag)
lo (loop flag)
qw (quadword data flag)
wa (write address flag)

Description

The fill memory command writes to a range or block of memory locations. The range is specified by first entering the starting address, followed by the ending address. The data to be written to the entire block can be entered by the user after the ending address, or it can be automatically generated by the command, depending on the state of the write address and negate data flags.

Example

In this example, the starting address (400003) will be truncated to the nearest quadword (400000) because the quadword data flag is on. The ending address (40004F) is truncated to 400048 for the same reason. Therefore, the last address written with the specified data is 400040. Similar truncations can be seen with the bm command.

```
SROM> qw
QW ON

SROM> fm
A> 400003
A> 40004F
D> 0123456789ABCDEF
```

```
SROM> bm
A> 3ffff7
A> 40005f
00000000.003ffff0: fff0ffff.fff0ff7d
00000000.003ffff8: fff0ffff.fff0ffff
00000000.00400000: 01234567.89abcdef
00000000.00400008: 01234567.89abcdef
00000000.00400010: 01234567.89abcdef
00000000.00400018: 01234567.89abcdef
00000000.00400020: 01234567.89abcdef
00000000.00400028: 01234567.89abcdef
00000000.00400030: 01234567.89abcdef
00000000.00400040: 01234567.89abcdef
00000000.00400048: 00000000.00000000
00000000.00400050: 00000000.00000000
```

The write address flag allows for unique data to be written to each memory location. This can be useful in debugging memory problems.

```
SROM> qw
QW OFF

SROM> wa
Wrt Addr ON

SROM> fm
A> 400000
A> 400020

SROM> bm
A> 400000
A> 400024
00000000.00400000:  00400000
00000000.00400004:  00400004
00000000.00400008:  00400008
00000000.0040000c:  0040000c
00000000.00400010:  00400010
00000000.00400014:  00400014
00000000.00400018:  00400018
00000000.0040001c:  0040001c
00000000.00400020:  89abcdef
```

Sometimes, it is useful to have alternating patterns written to adjacent memory locations. The negate data flag (!d) can be helpful then.

```
SROM> !d
Neg Data ON

SROM> fm
A> 400000
A> 400020
D> 0


SROM> bm
A> 400000
A> 400020
00000000.00400000: ffffffff
00000000.00400004: 00000000
00000000.00400008: ffffffff
00000000.0040000c: 00000000
00000000.00400010: ffffffff
00000000.00400014: 00000000
00000000.00400018: ffffffff
00000000.0040001c: 00000000
```

fr

The follow-with-read (fr) command toggles the follow-with-read flag, enabling or disabling the execution of a read operation after the last operation executed by a command.

Control Flags

fw (follow-with-write flag)

Description

The follow-with-read command is an advanced command whose use is required in certain situations where a second operation is needed to achieve a desired result, such as continuously creating a cache victim or having an I/O read operation interspersed between memory writes. Enabling the follow-with-read flag causes a prompt for an additional address from which the read operation is performed, unless the follow-with-write flag is also enabled. In this case, the additional address is used for the write operation and the read operation is performed from the first address.

The read operation takes place at the end of the command sequence, but after all read and write operations required by the current command have executed (including any write operations due to the follow-with-write flag being enabled). The data read may or may not be displayed, depending on the current command.

The default state is off.

Example

In the following example, a memory hierarchy composed of a 2MB direct-mapped, write-back, read-allocate cache and system memory is assumed. This may be the case in a 21064-based system where the Dcache is off and an external 2MB cache is on.

To continuously write the same address and guarantee that the data makes it to the system memory and is not simply cached, you must create a victim after each write operation. To create a victim, map the read address to the same cache index as the write operation.

For example, writing to address 0x600000 and reading from 0x800000 ejects the cached write of address 0x600000. From the system memory's point of view, a read of 0x800000 will be followed with a write of 0x600000, followed by another read of 0x800000 and so on until the loop is broken by pressing a key.

```
SROM> lo                ! Repeat command until a key is pressed.
Loop ON

SROM> fr                ! Follow deposit command with a read.
FollowRd ON

SROM> di                ! Don't display the data from the read.
Disp OFF

SROM> dm
A> 600000               ! Address of write
A> 800000               ! Address of read; a multiple of 2MB.
D> 0f0f0f0f             ! Data for the write.
                        ! (key pressed)
```

To test the data path and cache timing between the CPU and the external cache, provide the same address for the read as the write. For example, each write operation in the following fm command is followed with a read operation from the external cache. Note that the read address is also incremented by the same amount as the write address.

```
SROM> lo                ! Don't repeat fill memory command.
Loop OFF

SROM> di                ! Print the data from the reads.
Disp ON

SROM> wa                ! Use the address as the data for the writes.
Wrt Addr ON

SROM> fr                ! Follow deposit command with a read.
FollowRd ON
```

```
SROM> fm
A> 600000                  ! Starting address of fill memory command.
A> 600020                  ! Ending address of fill memory command.
A> 600000                  ! Address to begin reading from.
00000000.00600000: 00600000
00000000.00600004: 00600004
00000000.00600008: 00600008
00000000.0060000c: 0060000c
00000000.00600010: 00600010
00000000.00600014: 00600014
00000000.00600018: 00600018
00000000.0060001c: 0060001c
```

During the execution of the mt command, it may be desirable for a read operation to I/O space to follow each write and read operation in the test. This could detect potential problems in the memory controller where memory and I/O accesses are handled. This command does not print the data from the read operation to I/O space.

```
SROM> di                   ! Ignored by the memory test command.
Disp ON

SROM> wa                   ! Use the address as the data for the writes.
Wrt Addr ON

SROM> fr                   ! Follow deposit command with a read.
FollowRd ON

SROM> mt
A> 500000                  ! Start memory test at this address.
A> 600000                  ! End memory test at this address.
A> 3fff80000               ! Follow with reads beginning at this address.
```

If the follow-with-write flag is also enabled, the additional address requested is used for this write operation and the read operation is performed from the first address. In the following example, two blocks of memory are filled with their own addresses and read operations from the first block are displayed after each write operation to the first block.

```
SROM> di                   ! Ignored by the memory test command.
Disp ON

SROM> wa                   ! Use the address as the data for the writes.
Wrt Addr ON
```

```
SROM> fw                    ! Follow with a write (before the read).
FollowWr ON

SROM> fr                    ! Follow deposit command with a read.
FollowRd ON

SROM> fm
A> 500000                   ! Begin to write at this address
A> 500020                   ! and end with this one.
A> 600000                   ! Write to this address range after each
                            ! write from first block.
00000000.00500000: 00500000         ! Reads from first block.
00000000.00500004: 00500004
00000000.00500008: 00500008
00000000.0050000c: 0050000c
00000000.00500010: 00500010
00000000.00500014: 00500014
00000000.00500018: 00500018
00000000.0050001c: 0050001c
```

fw

The follow-with-write (fw) command toggles the follow-with-write flag, enabling or disabling the execution of a write operation after the last operation executed by a command.

Control Flags

Not applicable.

Description

The follow-with-write command is an advanced command whose use is required in certain situations where a second operation is needed to achieve a desired result, such as continuously creating a cache victim or having an I/O write operation interspersed between memory read operations. Enabling the follow-with-write flag causes a prompt for an additional address to which the write will be performed.

The write operation takes place at the end of the command sequence, but before any read operations due to the follow-with-read flag being enabled. The data used for the write operation varies. If the write operation follows a read operation, then the data used comes from the read operation. If the write operation follows a write operation, then it uses the data supplied by the user or the data generated by the command (see the wa and !d commands).

The default state is off.

Example

In the following example, the fm command writes the block beginning at 0x400000 with the specified data pattern, because the follow-with-write (fw) flag is enabled; it also writes the same data to the block starting 0x500000. For the bm command, the write operation gets its data from the previously executed read operations, in effect, making this sequence a copy command.

```
SROM> fw          ! Follow the last operation with a write.
FollowWr ON
```

```
SROM> fm
A> 400000          ! First block gets written with data pattern.
A> 400010
A> 500000          ! Second block also gets written with the same data pattern.
D> abcdef

SROM> bm
A> 500000          ! Read from this block.
A> 500010
A> 600000          ! Write to this block.
00000000.00500000: 00abcdef
00000000.00500004: 00abcdef
00000000.00500008: 00abcdef
00000000.0050000c: 00abcdef

SROM> fw
FollowWr OFF

SROM> bm           ! This shows that a copy function has taken place.
A> 600000
A> 600010
00000000.00600000: 00abcdef
00000000.00600004: 00abcdef
00000000.00600008: 00abcdef
00000000.0060000c: 00abcdef
```

lo

The loop (lo) command enables or disables looping or repeating of a command by toggling the loop flag.

Control Flags

Not applicable.

Description

When the loop flag is enabled, examine and deposit commands are repeated until you stop the looping by pressing a key. This feature is useful for hardware timing of read and write operations to the external cache or memory system, or anytime a command needs to be repeated continuously.

The default state is off.

Example

In the following example that shows the dm command, the data pattern is written to address 500000 continuously until a key is pressed. The em command reads and displays the data from address 500000 continuously until a keystroke is detected.

```
SROM> lo
Loop ON

SROM> dm
A> 500000
D> 12345678
                        ! (key pressed)
```

```
SROM> em
A> 500000
00000000.00500000: 12345678
00000000.00500000: 12345678
00000000.00500000: 12345678
00000000.00500000: 12345678
00000000.00500000: 12345678
00000000.00500000: 12345678
00000000.00500000: 12345678
00000000.00500000: 12345678
                     ! (key pressed)
```

The mt command continuously checks the range between 400000 and 500000.

```
SROM> wa
Wrt Addr ON

SROM> mt
A> 400000
A> 500000
                          !(key pressed)
```

mt

The memory test (mt) command performs a simple write-read-compare test.

Control Flags

!d (negate data flag)
ba (base address flag)
fr (follow-with-read flag)
fw (follow-with-write flag)
lo (loop flag)
qw (quadword data flag)
wa (write address flag)

Description

The memory test command first writes to the memory range specified with the user-specified data or command-generated data. It then begins reading back the data and comparing it against what was written. If a mismatch is encountered, both the read and the expected data are displayed on the screen.

Example

In the following example, the data pattern 12345678 is written to the block starting at 400000 and ending at 600000. The range is then read back and compared with the expected data 12345678. In this example, no mismatches are found.

```
SROM> mt
A> 400000
A> 600000
D> 12345678
```

In this next example, a bit stuck high is detected by the memory test.

```
SROM> wa                ! Use address of write as the data
Wrt Addr ON
```

```
SROM> mt
A> 400000
A> 500000
00000000.00400000:10400000   ! The data read back has an extra bit set.
Expect: 00400000
00000000.00400004:10400004
Expect: 00400004
00000000.00400008:10400008
Expect: 00400008
00000000.0040000c:1040000c
Expect: 0040000c
00000000.00400010:10400010
Expect: 00400010
00000000.00400014:10400014
Expect: 00400014
                             ! A key was pressed
```

pr

The print register (pr) command displays the contents of the general-purpose CPU registers.

Control Flags

Not applicable.

Description

The contents of the general-purpose CPU registers are saved to PALtemp registers when the mini-debugger is first entered. The pr command allows for their viewing. Note that because the dc command uses two of these PALtemp registers, the results in R1 and R2 will be changed after a dc command.

Note: This command may not be available in all standard SROMs supplied with the Alpha motherboards due to a lack of space. If this is the case, recompile the mini-debugger source files supplied in the Alpha SDK with the FULL_MDBG compile switch defined to enable this command.

Example

```
SROM> pr
R00:    00000000.00000000
R01:    00000000.00000006
R02:    000007f8.00000000
R03:    00000000.00000030
R04:    00000000.00000000
R05:    00000000.00000000
R06:    00000000.00300000
R07:    00000000.00001428
R08:    00000000.02000000
R09:    00000000.00000004
R10:    00000000.00026890
R11:    00000000.000262a0
R12:    00000000.00026560
R13:    0000004e.2001c665
R14:    00000000.00026590
R15:    00000000.000265a0
R16:    00000000.00026580
R17:    00000000.00008000
R18:    00000080.00000080
R19:    00000000.01ffdf30
R20:    ffffffff.fffdffff
R21:    00000000.00300000
R22:    0048484c.4c4e4e4e
R23:    00000000.0032edec
R24:    00000000.00000000
R25:    00000000.01ffdd25
R26:    00000000.00011fa8
R27:    00000000.00200080
R28:    00000000.0007a900
R29:    00000000.0007ce50
R30:    00000000.01ffdf30
```

qw

The quadword (qw) command enables or disables the quadword data flag.

Control Flags

Not applicable.

Description

The quadword command changes the state of the quadword data flag. When the quadword data flag is enabled, all operations are performed on 64-bit data. When off, only 32 bits of data are used.

The default state is off.

Example

In this example, the first bm command loads 32 bits of data using the ldl/p instruction from addresses 3.FFF80000, 3.FFF80004, 3.FFF80008, and 3.FFF8000C. After quadword mode is enabled, the same command performs only two 64-bit loads of data using the ldq/p instruction from addresses 3.FFF80000 and 3.FFF80008.

```
SROM> bm
A> 3fff80000
A> 3fff80010
00000003.fff80000: 5a5ac3c3
00000003.fff80004: a5a53c3c
00000003.fff80008: 00000038
00000003.fff8000c: 00006579

SROM> qw
QW ON

SROM> bm
A> 3fff80000
A> 3fff80010
00000003.fff80000: a5a53c3c.5a5ac3c3
00000003.fff80008: 00006579.00000038
```

rt

The return (rt) command exits the mini-debugger.

Control Flags

Not applicable.

Description

The return command exits the mini-debugger, returning to the calling program if there is one. For the standalone mini-debugger, it simply exits.

Returning from a mini-debugger built into the SROMs provided with the motherboards allows you to continue with the booting process, as if the mini-debugger had not been invoked.

The rt command is also useful during debugging of SROM code you may write. You can place calls to the mini-debugger throughout the SROM code to act as breakpoints. You can then examine the system and CPU state at those points and return to the SROM code to continue its execution.

For information on using the rt command during debugging, refer to the sources for the EB164 motherboard SROM included in the Alpha SDK.

Example

An example of the rt command follows:

```
SROM> rt
```

sb

The set base (sb) command defines the base address to be added to user-specified addresses when the base flag is enabled.

Control Flags

Not applicable.

Description

Using a base address is convenient when examining or depositing of the same address frame is needed (particularly when the high-order bits of the address must be set). The base address entered with the set base command is added to the addresses entered in any subsequent examine or deposit commands, saving the user some typing. This command always enables the base flag.

Example

To access a flash ROM part that responds to addresses 3.FFF80000 through 3.FFFFFFFF, it may be more convenient to enter its base address and work with offsets, rather than typing the absolute address every time.

```
SROM> sb
A> 3fff80000
00000003.fff80000
BaseAddr ON

SROM> bm
A> 0
A> 10
00000003.fff80000: 5a5ac3c3
00000003.fff80004: a5a53c3c
00000003.fff80008: 00000038
00000003.fff8000c: 00006579

SROM> ba
00000003.fff80000
BaseAddr OFF
```

```
SROM> em
A> 400000
00000000.00400000: 00400000

SROM> ba
00000003.fff80000
BaseAddr ON

SROM> bm
A> 30
A> 40
00000003.fff80030: 00000001
00000003.fff80034: 0000be8a
00000003.fff80038: 47ff041f
00000003.fff8003c: 47ff041f
```

st

The start image (st) command begins execution at a specified address.

Control Flags

Not applicable.

Description

The start image command transfers control to the code residing at the specified address. If the address does not contain executable code, then the machine hangs. You must recycle the power to start again.

Note: The Alpha Microprocessors SROM Mini-Debugger is stored in the instruction cache (Icache). An st command causes the Icache to be overwritten and eliminates your ability to return to the mini-debugger.

Example

The st command begins execution of the image uploaded.

```
SROM> xm
A> 400000
D> 10

SROM> st                ! Address to start execution at.
A> 400000
```

wa

The write address (wa) command enables or disables the write address flag.

Control Flags

!d (negate data flag)

Description

When the write address flag is enabled, write operations use the destination address as their data. This allows for unique values to be written to individual locations (a useful feature when debugging the memory subsystem). If the negate data flag is enabled, the one's complement of the address being written is used instead. Commands that request data to be written will not do so if this flag is enabled.

The default state is off; data used by deposit commands is provided by the user.

Example

In the following example, after enabling the write address flag, the region between 400000 and 400020 is filled with its own address in 32-bit increments. With qw mode on, deposits are performed in quadword increments, and the data written changes in 64-bit increments.

```
SROM>  wa
Wrt Addr ON

SROM> fm
A> 400000
A> 400020
```

```
SROM> bm
A> 400000
A> 400020
00000000.00400000: 00400000
00000000.00400004: 00400004
00000000.00400008: 00400008
00000000.0040000c: 0040000c
00000000.00400010: 00400010
00000000.00400014: 00400014
00000000.00400018: 00400018
00000000.0040001c: 0040001c

SROM> qw
QW ON

SROM> fm
A> 800000
A> 800030

SROM> bm
A> 800000
A> 800030
00000000.00800000: 00000000.00800000
00000000.00800008: 00000000.00800008
00000000.00800010: 00000000.00800010
00000000.00800018: 00000000.00800018
00000000.00800020: 00000000.00800020
```

With the negate data flag enabled, the complement of the address is used instead.

```
SROM> qw
QW OFF

SROM> !d
Neg Data ON

SROM> fm
A> 500000
A> 500020
```

```
SROM> bm
A> 500000
A> 500020
00000000.00500000: ffafffff
00000000.00500004: ffaffffb
00000000.00500008: ffaffff7
00000000.0050000c: ffaffff3
00000000.00500010: ffafffef
00000000.00500014: ffafffeb
00000000.00500018: ffafffe7
00000000.0050001c: ffafffe3
```

xb

The external boot (xb) command begins execution of uploaded image.

Control Flags

Not applicable.

Description

The external boot command is used after an image has been uploaded with the xm command. It adds the necessary Icache flush code before it transfers control to the code residing at the address specified in the xm command. If no xm command has been executed, then this command has no effect.

Example

The following example begins execution of the image uploaded:

```
SROM> xm
A> 400000
D> 10

SROM> xb                ! Execution of code at 400000 begins.
```

xm

The external image to memory (xm) command loads an image into memory.

Control Flags

Not applicable.

Description

The external image to memory command loads an external image into memory. The first input is the destination address, followed by the number of bytes to load. The number of bytes should be a multiple of 8 and the starting address should be quadword aligned. Normally, this command is used by an external utility called xload.exe (or the unix utility uload), which is connected to the motherboard running the mini-debugger.

Example

A user can upload an image by typing the starting address and the number of bytes to upload. In this example, 16 bytes are copied from the SROM port to address 400000.

```
SROM> xm
A> 400000
D> 10
```

A

Support, Products, and Documentation

If you need technical support, a *Digital Semiconductor Product Catalog*, or help deciding which documentation best meets your needs, visit the Digital Semiconductor World Wide Web Internet site:

http://www.digital.com/semiconductor

You can also call the Digital Semiconductor Information Line or the Digital Semiconductor Customer Technology Center. Please use the following information lines for support.

For documentation and general information:	
Digital Semiconductor Information Line	
United States and Canada:	1-800-332-2717
Outside North America:	1-510-490-4753
Electronic mail address:	semiconductor@digital.com

For technical support:	
Digital Semiconductor Customer Technology Center	
Phone (U.S. and international):	1-508-568-7474
Fax:	1-508-568-6698
Electronic mail address:	ctc@hlo.mts.dec.com

Digital Semiconductor Products

To order Alpha microprocessors and motherboards, contact your local distributor. The following tables list some of the semiconductor products available from Digital Semiconductor.

Chips	Order Number
Alpha 21164-600 Microprocessor	21164–MB
Alpha 21164-533 Microprocessor	21164–P8
Alpha 21164-466 Microprocessor	21164–IB

Motherboard kits include the motherboard and motherboard user's manual.

Motherboard Kits	Order Number
AlphaPC 164LX Motherboard Windows NT	21A04–C0
AlphaPC 164 Motherboard Windows NT	21A04–B0
AlphaPC 164 Motherboard DIGITAL UNIX	21A04–B2

Design kits include full documentation and schematics. They do not include evaluation boards or related hardware.

Design Kits	Order Number
AlphaPC 164 Motherboard Design Kit	QR–21A04–12

Digital Semiconductor Documentation

The following table lists some of the available Digital Semiconductor documentation.

Title	Order Number
Alpha AXP Architecture Reference Manual[1]	EY–T132E–DP
Digital Semiconductor 21164 Microprocessor Hardware Reference Manual	EC–QP99B–TE
Digital Semiconductor AlphaPC 164LX Motherboard Product Brief	EC–R2RZA–TE
Digital Semiconductor AlphaPC 164LX Motherboard User's Manual	EC–R2ZQA–TE

Title	Order Number
Digital Semiconductor AlphaPC 164LX Motherboard Technical Reference Manual	EC–R46WA–TE
Digital Semiconductor AlphaPC 164 Motherboard Product Brief	EC–QUQKC–TE
AlphaPC 164 Motherboard User's Manual	EC–QPG0B–TE
Digital Semiconductor AlphaPC 164 Motherboard Technical Reference Manual	EC–QPFYB–TE
Digital Semiconductor AlphaPC 164 Motherboard Design Kit Read Me First	EC–QPFZA–TE
Digital Semiconductor AlphaPC 164 DIGITAL UNIX Product Brief	EC–QZT6B–TE
AlphaPC 164 Motherboard DIGITAL UNIX User's Manual	EC–QZT5B–TE
Digital Semiconductor Alpha Motherboards Software Developer's Kit and Firmware Update V3.0 Product Brief	EC–QXQKB–TE
Digital Semiconductor Alpha Motherboards Software Developer's Kit and Firmware Update Read Me First	EC–QERSG–TE
Digital Semiconductor Alpha Microprocessors Motherboard Debug Monitor User's Guide	EC–QHUVE–TE
Digital Semiconductor Alpha Microprocessors Motherboard Software Design Tools User's Guide	EC–QHUWC–TE
Digital Semiconductor Alpha Microprocessors Motherboard Windows NT 3.51 and 4.0 Installation Guide	EC–QLUAG–TE
Digital Semiconductor PALcode for Alpha Microprocessors System Design Guide	EC–QFGLC–TE
Digital Semiconductor Alpha SRM Console for Alpha Microprocessor Motherboards User's Guide	EC–QK8DE–TE

[1] To purchase the *Alpha AXP Architecture Reference Manual*, contact your local distributor or call Butterworth-Heinemann (Digital Press) at 1-800-366-2665.

Index

A

ABOX_CTL
  default setting, 2–6
AlphaPC 64
  sample session, 2–10
Audience, vii

B

ba
  See base flag command
base flag command, 3–6
Baud rate, 2–6
BIU_CTL register, 2–7
block memory command, 3–8
bm
  See block memory command
boot, 2–5

C

cm
  See compare command
Command features, 3–1
Command summary, 3–2
Commands, 3–1
  base flag, 3–6
  block memory, 3–8
  compare, 3–9
  copy, 3–11
  deposit CPU register, 3–12
  deposit memory, 3–16
  display, 3–14
  examine CPU registers, 3–18
  examine memory, 3–21
  external boot, 3–47
  external image to memory, 3–48
  fill memory, 3–24
  flags, 3–22
  follow-with-read, 3–27
  follow-with-write, 3–31
  loop, 3–33
  memory test, 3–35
  negate data, 3–4
  print register, 3–37
  quadword, 3–39
  return, 3–40
  set base, 3–41
  start image, 3–43
  write address, 3–44
compare command, 3–9
Connecting
  to a dumb terminal, 2–2
  to a PC, 2–3
  to a serial port, 2–5
  to a system for DIGITAL UNIX, 2–4
  to a system for Windows NT 3.51, 2–3
  to a system for Windows NT 4.0, 2–4
Conventions of document, viii

copy command, 3–11
cp
See copy command

D

!d
See negate data command
dc
See deposit CPU register command
Debugging memory faults, 2–45
Default conditions, 2–6
deposit CPU register command, 3–12
deposit memory command, 3–16
di
See display command
DIGITAL UNIX system, 2–4
DIGITAL UNIX system, connecting to, 2–4
DIGITAL UNIX tip command, 2–5
display command, 3–14
dm
See deposit memory command
Document
audience, vii
conventions, viii
purpose, vii
Document structure, viii
Dumb terminal, 2–2

E

EB164
sample session, 2–21
EB64
address data, 2–7
sample log, 2–8
sample session, 2–7
EB64+
sample session, 2–10
EB66
sample session, 2–14
EB66+
sample session, 2–14
ec
See examine CPU registers command
em
See examine memory command
examine CPU registers command, 3–18
examine memory command, 3–21
external boot command, 3–47
external image to memory command, 3–48

F

Features, 1–1
command language, 3–1
hardware debug, 2–2
fill memory command, 3–24
fl
See flags command
flags command, 3–22
fm
See fill memory command
follow-with-read command, 3–27
follow-with-write command, 3–31
fr
See follow-with-read command
fw
See follow-with-write command

G

Getting started, 2–1

H

Hardware debug features, 2–2
Hardware requirements, 2–1

I

ICCSR, default setting, 2–6
Introduction, 1–1

L

lo
See loop command
load, 2–5
loop command, 3–33

M

Machine check handler, 2–45
MCHK
See machine check handler
memory test command, 3–35
mt
See memory test command

N

negate data command, 3–4

O

onnecting, 2–4

P

PC, connecting to, 2–3
pr
See print register command
print register command, 3–37
Purpose of document, vii

Q

quadword command, 3–39
qw
See quadword command

R

Required hardware, 2–1
return command, 3–40
rt
See return command
Running the mini-debugger, 2–6

S

Sample session
AlphaPC 164, 2–28
AlphaPC 64, 2–10
EB164, 2–21
EB64, 2–7
EB64+, 2–10
EB66, 2–14
EB66+, 2–14
sb
See set base command
Serial port
connecting to, 2–5
Serial port setup, 2–2
set base command, 3–41
Setting BIU_CTL register, 2–7
SROM serial port, 2–2
st
See start image command
start image command, 3–43
Starting the mini-debugger, 2–6
Structure of document, viii
Synchronization, 2–6

T

Table of commands, 3–2
Terminal, connecting to, 2–2
tip
DIGITAL UNIX command, 2–5

U

User commands, 3–3
User interface features, 3–1

W

wa
See write address command
Windows NT 3.51, 2–3
Windows NT 3.51 system, connecting to, 2–3

Windows NT 4.0, 2–4
Windows NT 4.0 system, connecting to, 2–4
write address command, 3–44

X

xb
See external boot command
xm
See external image to memory command